1 eclipse 使用和程序的断点调试

1.1 Eclipse 的使用

工作空间目录是纯英文不带空格的路径
在 eclipse 下 Java 程序的编写和运行，及 java 运行环境的配置。
新建 java 工程 day01，在弹出窗口中可配置 jre
工程右键属性可配置编辑器的版本

1.2 调试程序

1.3 Debug 窗口

Resume(F8)到下一个断点
Step into(F5)进入到函数等里面
Step over(F6)到下一行代码
Step return(F7)返回到调用的下一行
Drop to Frame 返回到当前方法的第一行，
Terminate (F12)终止虚拟机，程序就结束了。（调试完后用）
右键 watch 观察变量的值

1.4 Breakpoints 窗口

移除所以断点

1.5 断点注意问题

1. 调完后，移除所以断点
2. 调完后，一定要结束断点的 JVM。

2 eclipse 常用快捷键

MyEclipse 设置工作空间默认编码 utf-8 等，使新建工程使用默认编码
菜单栏——Window / Preferences / General / Workspace 。
内容提示：Alt + / Content Assist
选中多行代码，按 Tab 键是整块向右推进，按 Shift+Tab 是整块向左缩进
快速修复：Ctrl + 1
导包：Ctrl + shift + O
格式化代码块：ctrl + shift + F
向前向后：Alt + 方向键（left right arrow）查看源代码时
添加注释 Ctrl+Shift+/
除去注释 Ctrl+Shift+\
查看源代码 Ctrl+单击 ctrl+shift+t
查看方法说明：F2
重置透视图 Window menu 下

更改为大写 Ctrl+Shift+X
更改为小写 Ctrl+Shift+Y
复制行 Ctrl+Alt+向下键(有些不能用)
查看类的继承关系 Ctrl+T
查看快捷键 Ctrl+shift+L

3 junit 测试框架

在 Outline 窗口方法上右键 Run As /JUnit Test 测试某一个方法，类上右键 run as /JUnit Test 测试这个类的所有方法
1、用 junit 进行单元测试的时候，在每个被测试的方法中必须加上@Test 注解
2、用@Before 注解是在每个被测试的方法前执行。
3、用@After 注解是在每个被测试的方法后执行。
4、用@BeforeClass 注解的静态方法是在所有方法被测试之前执行的方法，就像类里面的构造方法一样。用来初始化一些要用到的变量等资源。
5、用@AterClass 注解的静态方法是在所有被测试的方法之后执行。相当于 c++中析构函数。用来释放一些资源。
6、使用断言类 Assert 可以判断被测试的方法的返回值是否跟你预期的相同。

```
//person.java
package cn.itcast.elclipse;

public class Person {
	public void eat()
	{
		System.out.println("eating.");
	}
	public String run()
	{
		System.out.println("runing.");
		return "1";
	}
}
```

```
//demo4.java
package cn.itcast.elclipse;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class Demo4 {
	@Before
	public void before(){
		System.out.println("before");
	}

	@Test
	public void testRun()
	{
		Person p=new Person();
		p.run();
	}
	@Test
	public void testEat(){
		Person p=new Person();
		p.eat();
	}
	@After
	public void after(){
		System.out.println("after");
	}
}
```

```
//Demo5.java
package cn.itcast.elclipse;

import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;

public class Demo5 {
	@BeforeClass
	public void beforeClass(){
		System.out.println("beforeclass");
	}

	@Test
	public void testRun()
	{
		Person p=new Person();
		p.run();
	}
	@Test
	public void testEat(){
		Person p=new Person();
		p.eat();
	}
	@AfterClass
	public void afterClass(){
		System.out.println("afterclass");
	}
}
```

```
//Demo6.java
package cn.itcast.elclipse;

import junit.framework.Assert;

import org.junit.Test;

public class Demo6 {

	@Test
	public void testRun()
	{
		Person p=new Person();
		Assert.assertEquals("2", p.run());
	}
}
```

4 java5 的静态导入和自动装箱拆箱.avi

JDK5 中新增了很多新的 java 特性，利用这些新语法可以帮助开发人员编写出更加高效、清晰，安全的代码。
静态导入　自动装箱/拆箱　增强 for 循环　可变参数　枚举反射内省　泛型　元数据

4.1 静态导入

JDK 1.5 增加的静态导入语法用于导入类的某个静态属性或方法。使用静态导入可以简化程序对类静态属性和方法的调用。
语法：
Import static 包名.类名.静态属性|静态方法|*
例如：

4.2 自动装箱/拆箱

JDK5.0 的语法允许开发人员把一个基本数据类型直接赋给对应的包装类变量，或者赋给 Object 类型的变量，这个

过程称之为自动装箱。
自动拆箱与自动装箱与之相反，即把包装类对象直接赋给一个对应的基本类型变量。
典型应用：

静态导入	自动装箱/拆箱

import static java.lang.System.out
import static java.lang.Math.*
//Demo1
package cn.itcast.demo;
import static java.lang.System.out;
import static java.util.Arrays.*;
 	
public class Demo1 {
public static void main(String[] args) {
out.print("main");
int []a=new int[]{6,5,3};
sort(a);
for(int i:a)
out.print(i);
}
}
// main356
	package cn.itcast.demo;
	
	import java.util.ArrayList;
	import java.util.Iterator;
	import java.util.List;
	
	public class Demo2 {
	public static void main(String[] args) {
	Integer i=1;//装箱
	int j=i;//拆箱
	//典型应用
	List list = new ArrayList();
	list.add(1);
	int k = (Integer)list.get(0);
	Iterator it=list.iterator();
	while(it.hasNext())
	{
	int m=(Integer)it.next();//拆箱
	}
	}
	}

5 增强 for 循环

5.1 增强 for 循环

引入增强 for 循环的原因：在 JDK5 以前的版本中，遍历数组或集合中的元素，需先获得数组的长度或集合的迭代器，比较麻烦！
因此 JDK5 中定义了一种新的语法——增强 for 循环，以简化此类操作。增强 for 循环只能用在**数组、或实现 Iterator 接口的集合类**上
语法格式：

for(变量类型变量　：需迭代的数组或集合){

}

Map map=new HashMap();
//Map map2=new LinkedHashMap<K, V>();
map.put("1", "aaa");
map.put("2", "bbb");
map.put("3", "ccc");	

//传统方式 1	//增强 for 循环的 1 种方式

Set set=map.keySet();
Iterator it=set.iterator();
while(it.hasNext()){
String key=(String)it.next();
String value=(String) map.get(key);
 	
System.out.println("key="+key+",value="+value);
}
	for(Object obj:map.keySet()){
	String key2=(String)obj;
	String value2=(String)map.get(key2);
	System.out.println("key2="+key2+",value2="+value2);
	}

//传统方式 2
Set set2=map.entrySet();
Iterator it2=set2.iterator();
while(it2.hasNext()){
Map.Entry entry=(Entry)it2.next();
 	
System.out.println("key="+entry.getKey()+",value="+entry.getValue());
}
	//增强 for 循环的 2 种方式
	for(Object obj:map.entrySet()){
	Map.Entry entry3=(Entry) obj;
	String key3=(String) entry3.getKey();
	String value3=(String) entry3.getValue();
	
	System.out.println("key3="+key3+",value3="+value3);
	}

	//增强 for 循环需要注意的问题：只适合取数据
	int arr[]={1,2,3};
	for(int i: arr){
	i=10;
	}
	System.out.println(arr[0]); // 1
	List li=new ArrayList();
	li.add("1");
	for(Object obj : li){
	obj="888";
	}
	System.out.println(li.get(0));// 1

6可变参数(相当于动态的数组)

测试 JDK 中具有可变参数的类 Arrays.asList()方法。分别传多个参数、传数组，传数组又传参的情况。
注意：传入基本数据类型数组的问题。
从 JDK 5 开始, Java 允许为方法定义长度可变的参数。语法：

```
public void foo(int … args){
}
```

注意事项：
调用可变参数的方法时，编译器将自动创建一个数组保存传递给方法的可变参数，因此，程序员可以在方法体中以数组的形式访问可变参数
可变参数只能处于参数列表的最后，所以一个方法最多只能有一个长度可变的参数

public void testSum(){ sum(1,2,3,4);	public void bb(){ //public static <T> List<T>asList(T... a)

```
        int arr[]={5,6,7};
        sum(arr);
    }
    public void sum(int ...nums){
        //可变参数当成数组
        int sum=0;
        for(int i:nums){
            sum+=i;
        }
        System.out.println(sum);
    }
```

```
        List list=Arrays.asList("1","2","3");
        System.out.println(list);//[1, 2, 3]

        String arr[]={"1","2","3","4"};
        list=Arrays.asList(arr);
        System.out.println(list);//[1, 2, 3, 4]

        int nums[]={1,2,3,4,5};
        list=Arrays.asList(nums);//存储只是一个对象
        System.out.println(list);//[[I@120d62b]

        Integer nums2[]={1,2,3,4,5};
        list=Arrays.asList(nums2);
        System.out.println(list);//[1, 2, 3, 4, 5]

    }
```

```
//可变参数注意的问题
    //public void aa(int ...nums,int s){}//不可以
    //public void bb(int s ,int ...nums)//可以
```

7枚举类

为什么需要枚举？
一些方法在运行时，它需要的数据不能是任意的，而必须是一定范围内的值，此类问题在 JDK5 以前采用自定义带有枚举功能的类解决，Java5 以后可以直接使用枚举予以解决。
JDK 5 新增的 enum 关键字用于定义一个枚举类。

7.1 枚举类

枚举类具有如下特性：

1.枚举类也是一种特殊形式的 Java 类。
2.枚举类中声明的每一个枚举值代表枚举类的一个实例对象。A，B，C，D
3.与 java 中的普通类一样，在声明枚举类时，也可以声明属性、方法和构造函数，但枚举类的构造函数必须为私有的（private 这点不难理解）。为什么？防止其他类初始化对象
4.枚举类也可以实现接口、或继承抽象类。
5.JDK5 中扩展了 swith 语句，它除了可以接收 int, byte, char, short 外，还可以接收一个枚举类型。
6.若枚举类只有一个枚举值，则可以当作单态设计模式使用。
7.Java 中声明的枚举类，均是 java.lang.Enum 类的孩子，它继承了 Enum 类的所有方法。常用方法：
name()
ordinal()//返回枚举常量的序数（它在枚举声明中的位置，其中初始常量序数为零）。

valueof(Class enumClass, String name) //返回带指定名称的指定枚举类型的枚举常量。

//String str=”B”;Grade g=Grade.valueOf(Grade.class,str);

values() 此方法虽然在 JDK 文档中查找不到，但每个枚举类都具有该方法，它遍历枚举类的所有枚举值非常方便。

练习：请编写一个关于星期几的枚举 WeekDay，要求：
枚举值：Mon、Tue 、Wed 、Thu 、Fri 、Sat 、Sun
该枚举要有一个方法，调用该方法返回中文格式的星期。

```
package cn.itcast.enumeration;
import org.junit.Test;
public class Demo1 {
	@Test
	public void test() {
		print(Grade.B);
	}

	public void print(Grade g) // A B C D E
	{
		String value=g.getValue();
		System.out.println(value);
	}
}

/*
 * class Grade{ private Grade(){ }
 *   public static final Grade A=new Grade();
 *   public static final Grade B=new Grade();
 *   public static final Grade C=new Grade();
 *   public static final Grade D=new Grade();
 *   public static final Grade E=new Grade();
 *   }
 */
//如何定义枚举的构造函数、方法、字段
enum Grade {// class A 100-90 B 89-80 C 79-70 D 69-60 E 59-0
	A("100-90"), B("89-80"), C("79-70"), D("69-60"), E("59-0");// object
	private String value;
	private Grade(String value){
		this.value=value;
	}
	public String getValue(){
		return this.value;
	}
}
```

```
package cn.itcast.enumeration2;
import org.junit.Test;
public class Demo1 {
	@Test
	public void test() {
		print(Grade.B);	//89-80,良
	}

	public void print(Grade g) // A B C D E
	{
		String value=g.getValue();
		String value2=g.localeValue();

		System.out.println(value+","+value2);
	}
}

//带抽象方法的枚举
enum Grade {
// class A 100-90 优 B 89-80 良 C 79-70 一般 D 69-60 差 E 59-0 不及格
	//A.b。C。D 就是对象
A("100-90"){
		public String localeValue(){
			return "优";
		}
	},
	B("89-80"){
		public String localeValue(){
			return "良";
		}
	},
	C("79-70"){
		public String localeValue(){
			return "一般";
		}
	},
	D("69-60"){
```

实际开法中，把客服端传过来的 sex 字符串，转换成枚举对象
 	
Grade g=Grade.valueOf(str);
 	
Grade gs[]=Grade.values();
For(Grade)
	public String localeValue(){
	return "差";
	}
	},
	E("59-0"){
	public String localeValue(){
	return "不及格";
	}
	};// object
	
	private String value;//封装每个对象对应的分数
	private Grade(String value){
	this.value=value;
	}
	public String getValue(){
	return this.value;
	}
	public abstract String localeValue();
	}

8反射技术

8.1 反射什么—Class 类

1. 反射就是把 Java 类中的各种成分映射成一个个的 java 对象。例如，一个类有：成员变量，方法，构造方法，包等等信息，利用反射技术可以对一个类进行解剖，把各个组成部分映射成一个个对象。
2. Class 类用于表示.class 文件，画图演示一个对象的创建过程。
3. 如何得到某个 class 文件对应的 class 对象。

类名.class，　对象.getClass()　Class.forName(“类名”)
以上三种方式，JVM 都会把相应 class 文件装载到一个 class 对象中(只装载一次)
创建类的实例：Class.newInstance 方法
数组类型的 Class 实例对象
Class.isArray()
总之，只要是在源程序中出现的类型，都有各自的 Class 实例对象，例如，int，void…
反射就是把 Java 类中的各种成分映射成一个个的 java 对象。例如，一个类有：成员变量，方法，构造方法，包等等信息，利用反射技术可以对一个类进行解剖，把各个组成部分映射成一个个对象。
掌握反射技术的要点在于：**如何从一个 class 中反射出各个组成部分。反射出来的对象怎么用**。

8.2 Constructor 类

（1）Constructor 类代表某个类中的一个构造方法
得到某个类所有的构造方法：
例子：Constructor [] constructors= Class.forName("java.lang.String").getConstructors();
得到某一个构造方法：

例子：　　//获得方法时要用到类型

Constructor constructor = Class.forName(“java.lang.String”).getConstructor(StringBuffer.class);

（2）创建实例对象：

通常方式：String str = new String(new StringBuffer("abc"));

反射方式：　String str = (String)constructor.newInstance(new StringBuffer("abc"));

Class.newInstance()方法：

例子：

String obj = (String)Class.forName("java.lang.String").newInstance();

该方法内部先得到默认的构造方法，然后用该构造方法创建实例对象。

8.3 Field 类

Field 类代表某个类中的一个成员变量

问题：得到的 Field 对象是对应到类上面的成员变量，还是对应到对象上的成员变量？类只有一个，而该类的实例对象有多个，如果是与对象关联，哪关联的是哪个对象呢？所以字段 fieldX 代表的是 x 的定义，而不是具体的 x 变量。(注意访问权限的问题)

示例代码：

```
ReflectPoint point = new ReflectPoint(1,7);
Field y = Class.forName("cn.itcast.corejava.ReflectPoint").getField("y");
System.out.println(y.get(point));
//Field x = Class.forName("cn.itcast.corejava.ReflectPoint").getField("x");
Field x = Class.forName("cn.itcast.corejava.ReflectPoint").getDeclaredField("x");
x.setAccessible(true);
System.out.println(x.get(point));
```

8.4 Method 类

Method 类代表某个类中的一个成员方法

得到类中的某一个方法：

例子：　　Method charAt = Class.forName("java.lang.String").getMethod("charAt", int.class);

调用方法：

通常方式：System.out.println(str.charAt(1));

反射方式：System.out.println(charAt.invoke(str, 1));

如果传递给 Method 对象的 invoke()方法的第一个参数为 null，这有着什么样的意义呢？说明该 Method 对象对应的是一个静态方法！

jdk1.4 和 jdk1.5 的 invoke 方法的区别：

Jdk1.5：public Object invoke(Object obj,Object... args)

Jdk1.4：public Object invoke(Object obj,Object[] args)，即按 jdk1.4 的语法，需要将一个数组作为参数传递给 invoke 方法时，数组中的每个元素分别对应被调用方法中的一个参数，所以，调用 charAt 方法的代码也可以用 Jdk1.4 改写为 charAt.invoke(“str”, new Object[]{1})形式。

8.5 用反射方式反射类中的 main 方法

目标：写一个程序，这个程序能够根据用户提供的类名，去执行该类中的 main 方法。用普通方式调完后，大家要明白为什么要用反射方式去调啊？

问题：

启动 Java 程序的 main 方法的参数是一个字符串数组，即 public static void main(String[] args)，通过反射方式来调用这个 main 方法时，如何为 invoke 方法传递参数呢？按 jdk1.5 的语法，整个数组是一个参数，而按 jdk1.4 的语法，数组中的每个元素对应一个参数，当把一个字符串数组作为参数传递给 invoke 方法时，javac 会到底按照哪种

语法进行处理呢？jdk1.5 肯定要兼容 jdk1.4 的语法，会按 jdk1.4 的语法进行处理，即把数组打散成为若干个单独的参数。所以，在给 main 方法传递参数时，不能使用代码 mainMethod.invoke(null,new String[]{“xxx”})，javac 只把它当作 jdk1.4 的语法进行理解，而不把它当作 jdk1.5 的语法解释，因此会出现参数类型不对的问题。

解决办法：

mainMethod.invoke(null,new Object[]{new String[]{"xxx"}});

mainMethod.invoke(null,(Object)new String[]{"xxx"}); ，编译器会作特殊处理，编译时不把参数当作数组看待，也就不会数组打散成若干个参数了

```
package cn.itcast.reflect;
import java.io.InputStream;
import java.util.List;
public class Person {

	public String name="aaaa";
	private int password;
	private static int age=23;

	public Person(){
		System.out.println("person");
	}
	public Person(String name){
		this.name=name;
		System.out.println("person name");
	}
	public Person(String name,int password){
		this.name=name;
		System.out.println("person name
password");
	}
	private Person(List list){
		System.out.println("list");
	}

	public void aa1(){
		System.out.println("aa1");
	}
	public void aa1(String name,int
password){
		System.out.println("name=
"+name+" password="+password);
	}
	public Class[] aa1(String name,int[]
password){
		return new Class[]{String.class};
	}
	private void aa1(InputStream in){
		System.out.println(in);
	}
```

```
package cn.itcast.reflect;
import java.lang.reflect.Field;
import org.junit.Test;

//反射类的字段
public class Demo4 {
	//public String name="aaaa";
	@Test
	public void test1() throws Exception{
		Class clazz=Class.forName("cn.itcast.reflect.Person");
		Field f=clazz.getField("name");
		Person p=new Person();
		Object value= f.get(p);//获取字段值
		Class type=f.getType();//字段类型
		System.out.println(type);//class java.lang.String

if(type.equals(String.class)){
			String S_value=(String)value;
			System.out.println(S_value);//aaaa
		}

		//设置值
		f.set(p, "ppppp");
		System.out.println(f.get(p));//ppppp
		System.out.println(String.class);//class java.lang.String

	}
	//private int password;
	@Test
	public void test2() throws Exception{
		Class clazz=Class.forName("cn.itcast.reflect.Person");
		Field f=clazz.getDeclaredField("password");
		f.setAccessible(true);
		Person p=new Person();
		f.set(p, 123);
		System.out.println(f.get(p));//123
	}
	//private static int age=23;
	@Test
	public void test3() throws Exception{
```

```
    public static void aa1(int num){
        System.out.println(num);
    }
    public static void main(String []args){
        System.out.println("main");
    }

}
```

```
        Class clazz=Class.forName("cn.itcast.reflect.Person");
        Field f=clazz.getDeclaredField("age");
        f.setAccessible(true);
        System.out.println(f.get(null));//23
    }
}
```

```
package cn.itcast.reflect;

import java.lang.reflect.Constructor;
import java.util.ArrayList;
import java.util.List;

import org.junit.Test;

//反射类的构造函数，创建类的对象
public class Demo2 {




//public Person()
    @Test
    public void test1() throws Exception{

//加载.class 类文件到内存中
Class clazz=Class.forName("cn.itcast.reflect.Person");
        Constructor c=clazz.getConstructor(null);
        Person p=(Person) c.newInstance(null);//person
        System.out.println(p.name);//aaaa
    }


//public Person(String name)
    @Test
    public void test2() throws Exception{
        Class
clazz=Class.forName("cn.itcast.reflect.Person");
        Constructor c=clazz.getConstructor(String.class);
        Person p=(Person) c.newInstance("abc");//person
name
        System.out.println(p.name);//abc
    }
```

```
package cn.itcast.reflect;

import java.io.FileInputStream;
import java.io.InputStream;
import java.lang.reflect.Method;

import org.junit.Test;
//反射类的方法
public class Demo3 {




//    public void aa1()
    @Test
    public void test1() throws Exception{
        Class clazz=Class.forName("cn.itcast.reflect.Person");
        Method method=clazz.getMethod("aa1", null);
        Person p=new Person();//person
        method.invoke(p, null);//调用谁的对象的方
法;方法所传递的参数
    }



//public void aa1(String name,int password)
    @Test
    public void test2() throws Exception{
        Class clazz=Class.forName("cn.itcast.reflect.Person");
        Method                    method=clazz.getMethod("aa1",
String.class,int.class);
        Person p=new Person();//person
```

```
    //public Person(String name,int password)
    @Test
    public void test3() throws Exception{
        Class clazz=Class.forName("cn.itcast.reflect.Person");
        Constructor c=clazz.getConstructor(String.class,int.class);
        Person p=(Person) c.newInstance("abc",999);//person name password
        System.out.println(p.name);//abc
    }

//private Person(List list)
    @Test
    public void test4() throws Exception{
        Class clazz=Class.forName("cn.itcast.reflect.Person");
        Constructor c=clazz.getDeclaredConstructor(List.class);
        c.setAccessible(true);//暴力反射，统统打开访问权限
        Person p=(Person) c.newInstance(new ArrayList());//list
        System.out.println(p.name);//aaaa
    }

    //创建对象的另外一种途径，无参构造方法，等效test1
    public void test5() throws Exception{
        Class clazz=Class.forName("cn.itcast.reflect.Person");
        Person p=(Person) clazz.newInstance();
        System.out.println(p.name);//aaaa
    }
}
```

```
        method.invoke(p, "xxxx",99);//name= xxxx password=99
    }

    //public Class[] aa1(String name,int[] password)
    @Test
    public void test3() throws Exception{
        Class clazz=Class.forName("cn.itcast.reflect.Person");
        Method method=clazz.getMethod("aa1", String.class,int[].class);
        Person p=new Person();//person
        Class cs[]=(Class[]) method.invoke(p, "xxxx",new int[]{1,2,3});//name= xxxx password=99
        System.out.println(cs[0]);//class java.lang.String
    }

    //private void aa1(InputStream in)
    public void test4() throws Exception{
        Class clazz=Class.forName("cn.itcast.reflect.Person");
        Method method=clazz.getDeclaredMethod("aa1",InputStream.class);
        method.setAccessible(true);
        Person p=new Person();//person
        method.invoke(p,new FileInputStream("C:\\1.txt"));//
    }

    //public static void aa1(int num)
    public void test5() throws Exception{
        Class clazz=Class.forName("cn.itcast.reflect.Person");
        Method method=clazz.getMethod("aa1",int.class);
        method.invoke(null,777);//777  静态方法调用不需要对象，给对象也行
    }

//  public static void main(String []args){}//重点
    public void test6() throws Exception{
        Class clazz=Class.forName("cn.itcast.reflect.Person");
        Method method=clazz.getMethod("main",String [].class);
```

	//method.invoke(null,new String[]{"a","c"});//Wrong
	//method.invoke(null,"a","c");//Wrong
	String []str={"x","y","z"};
	method.invoke(null, (Object)str);
	method.invoke(null,newObject[]{new String[]{"a","c"} });
	method.invoke(null, (Object)new String[]{"a","c"});
	
	//jdk1.5 Method.invoke(Objet objecr,Object...args);用可变参数传递多个参数
	//jdk.1.4 Method.invoke(Obeject objecr ,Object obj[]{"xx","yy"});用数组传递多个参数
	
	//java 需要兼容 jdk1.4，就会拆除数组里的元素为多个参数，解决思路外面再加一个数组 new Object[]{new String[]{}}, 或者(Object)new String[]{"a","c"}欺骗系统不是数组；
	//开发中要注意传递数组的方法
	}
	}

9内省技术

9.1 用内省技术反省 javaBean

（1）内省(Introspector) — JavaBean

Introspector 类为通过工具学习有关受目标 Java Bean 支持的属性、事件和方法的知识提供了一个标准方法。

（2）什么是 JavaBean 和属性的读写方法?

有 get 或 set 方法就是一个属性，另外所有类继承了 Object 类的 getClass（）方法，所以还有一个属性 class。

（3）访问 JavaBean 属性的两种方式：

1.直接调用 bean 的 setXXX 或 getXXX 方法。

2.通过内省技术访问(java.beans 包提供了内省的 API)，内省技术访问也提供了两种方式。

A.通过 PropertyDescriptor 类操作 Bean 的属性

B. 通过 Introspector 类获得 Bean 对象的 BeanInfo，然后通过 BeanInfo 来获取属性的描述器（ PropertyDescriptor ），通过这个属性描述器就可以获取某个属性对应的 getter/setter 方法，然后通过反射机制来调用这些方法。

（4）什么情况下用内省？

package cn.itcast.introspector;
//该类共有 5 个属性
public class Person {
private String name;
private String password;
private int age;
public void
	package cn.itcast.introspector;
	
	import java.beans.BeanInfo;
	import java.beans.IntrospectionException;
	import java.beans.Introspector;
	import java.beans.PropertyDescriptor;
	import java.lang.reflect.InvocationTargetException;
	import java.lang.reflect.Method;

```
setAb(int a){

    }
    public String getName() {
        return name;
    }
    public String getPassword() {
        return password;
    }
    public int getAge() {
        return age;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public void setAge(int age) {
        this.age = age;
    }

}
```

```
import org.junit.Test;

public class Demo1 {

//得到 bean 所有属性
    Public void test1() throws IntrospectionException{
        BeanInfo info=Introspector.getBeanInfo(Person.class);

//去掉 Object 里的属性 getclass（）
BeanInfo info2=Introspector.getBeanInfo(Person.class,Object.class);
    PropertyDescriptor[] pds=info.getPropertyDescriptors();
        for(PropertyDescriptor pd:pds){
            System.out.println(pd.getName());
            //ab age class name    password
        }
    }

//操纵 bean 的指定属性：age
    @Test
    public void test2() throws  Exception{
        Person p=new Person();
        PropertyDescriptor pd=new PropertyDescriptor("age", Person.class);
        //得到属性的写方法，为属性赋值
        Method method=pd.getWriteMethod();
        method.invoke(p, 45);
        System.out.println(p.getAge());//45

        //获取属性的值
        method=pd.getReadMethod();
        System.out.println(method.invoke(p, null));//45
    }

//高级内容，获取当前操作的属性的类型
    @Test
    public void test3() throws  Exception{
        Person p=new Person();
        PropertyDescriptor pd=new PropertyDescriptor("age", Person.class);

        //得到属性的写方法，为属性赋值
        Method method=pd.getWriteMethod();
        System.out.println(pd.getPropertyType());//int
        method.invoke(p, 45);
        System.out.println(p.getAge());//45

        //获取属性的值
        method=pd.getReadMethod();
        System.out.println(method.invoke(p, null));//45
```

	} }

9.2 内省—beanutils 工具包

（1）Apache 组织开发了一套用于操作 JavaBean 的 API，这套 API 考虑到了很多实际开发中的应用场景，因此在实际开发中很多程序员使用这套 API 操作 JavaBean，以简化程序代码的编写。

在工程下新建 lib 目录，导入 commons-beanutils-1.8.3.jar 和支持包 commons-logging-1.1.1.jar

选中两个包，右键 build path/add to build path

（2）Beanutils 工具包的常用类：

BeanUtils

PropertyUtils

ConvertUtils.regsiter(Converter convert, Class clazz)

自定义转换器

package cn.itcast.beanutils;
 	
import java.util.Date;
 	
public class Person {
private String name;
private String password;
private int age;
private Date birthday;
public Date getBirthday() {
return birthday;
}
public void setBirthday(Date birthday) {
this.birthday = birthday;
}
public String getName() {
return name;
}
public String getPassword() {
return password;
}
public int getAge() {
return age;
}
public void setName(String name) {
this.name = name;
}
public void setPassword(String password) {
this.password = password;
	package cn.itcast.beanutils;
	
	import java.lang.reflect.InvocationTargetException;
	import java.text.ParseException;
	import java.text.SimpleDateFormat;
	import java.util.Date;
	import java.util.HashMap;
	import java.util.Locale;
	import java.util.Map;
	
	import org.apache.commons.beanutils.BeanUtils;
	import org.apache.commons.beanutils.ConversionException;
	import org.apache.commons.beanutils.ConvertUtils;
	import org.apache.commons.beanutils.Converter;
	import
	org.apache.commons.beanutils.locale.converters.DateLocaleConverter;
	import org.junit.Test;
	
	//使用 beanUtils 操纵 bean 的属性 (第三方）
	public class Demo1 {
	@Test
	public void test1() throws Exception{
	Person p=new Person();
	BeanUtils.setProperty(p, "age", 456);
	System.out.println(p.getAge());//456
	}
	@Test
	public void test2() throws Exception{
	String name="aaaa";
	String age="123";
	String password="pw";

```
    }
    public void setAge(int age) {
        this.age = age;
    }

}
```

所有的类都继承 OjectI 类，所以每个实体类都有+1getClass（）；

```
        Person p=new Person();
        //支持 8 种基本类型自动转换
        BeanUtils.setProperty(p, "name", name);
        BeanUtils.setProperty(p, "age", age);//string 自动转换 int
        BeanUtils.setProperty(p, "password", password);

        System.out.println(p.getName());//aaaa
        System.out.println(p.getAge());//123
        System.out.println(p.getPassword());//pw
    }
    @Test
    public void test3() throws Exception{
        String birthday="1983-12-1";
        //为了让日期赋值到 bean 的 birthday 属性上，给 beanUtils 注册一个日期转换器
        //ConvertUtils.register(converter, clazz);
            ConvertUtils.register(new Converter(){
            public Object convert(Class type, Object value) {
                if(value==null) return null;
                if(!(value instanceof String)){
                    throw new ConversionException(" 只支持 String 类型的转换");
                }
                String str=(String)value;
                if(str.trim().equals("")) return null;
                SimpleDateFormat df=new SimpleDateFormat("yyyy-MM-dd",Locale.US);
                try {
                    return df.parse(str);
                } catch (ParseException e) {
                    throw new RuntimeException(e);
                }
            }
        }, Date.class);
        Person p=new Person();
        BeanUtils.setProperty(p, "birthday", birthday);
        System.out.println(p.getBirthday());//pw
        System.out.println("___"+BeanUtils.getProperty(p, "birthday"));
    }

public void test5() throws Exception {
        Map map=new HashMap();
        map.put("name", "aaa");
        map.put("password", "123");
        map.put("brithday", "1980-09-09");
        ConvertUtils.register(new DateLocaleConverter(), Date.class);
```

```
        Person p=new Person();
    //用 map 集合填充 bean 属性,map 关键字和 bean 属性要一致
        BeanUtils.populate(p, map);
    System.out.println(bean.getName());
    }
}
```

10 泛型技术(Generic)

10.1 泛形的作用

（1）JDK5 以前，对象保存到集合中就会失去其特性，取出时通常要程序员手工进行类型的强制转换，这样不可避免就会引发程序的一些安全性问题。例如：

```
ArrayList list = new ArrayList();
list.add("abc");
Integer num = (Integer) list.get(0);  //运行时会出错，但编码时发现不了

list.add(new Random());
list.add(new ArrayList());
for(int i=0;i<list.size();i++){
    (?)list.get(i);              //此处取出来的对象应转换成什么类型
}
```

（2）JDK5 中的泛形允许程序员在编写集合代码时，就限制集合的处理类型，从而把原来程序运行时可能发生问题，转变为编译时的问题，以此提高程序的可读性和稳定性(尤其在大型程序中更为突出)。

注意：泛型是提供给 javac 编译器使用的，它用于限定集合的输入类型，让编译器在源代码级别上，即挡住向集合中插入非法数据。但编译器编译完带有泛形的 java 程序后，生成的 class 文件中将不再带有泛形信息，以此使程序运行效率不受到影响，这个过程称之为“擦除”。

泛形的基本术语，以 ArrayList<E>为例：<>念着 typeof

ArrayList<E>中的 E 称为类型参数变量

ArrayList<Integer>中的 Integer 称为实际类型参数

整个称为 ArrayList<E>泛型类型

整个 ArrayList<Integer>称为参数化的类型 ParameterizedType

10.2 泛型典型应用

使用迭代器迭代泛形集合中的元素。

使用增强 for 循环迭代泛形集合中的元素。

存取 HashMap 中的元素。

使用泛形时的几个常见问题：

使用泛形时，泛形类型须为引用类型，不能是基本数据类型

//使用泛型时，如果两边都使用到泛型，两边必须一样

```
ArrayList<String> list = new ArrayList<Object>(); //bad
ArrayList<Object> list = new ArrayList<String>(); //bad
ArrayList<String> list = new ArrayList ();//ok
ArrayList list = new ArrayList<String>();//ok
```

```
package cn.itcast.generic;

import java.util.ArrayList;
import java.util.List;

import org.junit.Test;

public class Demo1 {
	@Test
	public void test1(){
		List list=new ArrayList();
		list.add("111");
		list.add("222");
		list.add("333");
		//传统方式手工转换
		String i=(String) list.get(0);
		//下面注释代码手工转换编辑不报错，运行错误
		//Integer ii=(Integer) list.get(0);
		System.out.println(i);
	}
	@Test
	public void test2(){
		List <String>list=new ArrayList<String>();
		list.add("111");
		list.add("222");
		list.add("333");
		//现在不需要强制转换
		String i=list.get(0);
		//下面注释代码编译通不过
		//Integer ii=(Integer) list.get(0);
		System.out.println(i);
	}
}
```

```
package cn.itcast.generic;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Set;

import org.junit.Test;

public class Demo2 {
	@Test
	public void test1(){
		List <String>list=new ArrayList<String>();
		list.add("111");
		list.add("222");
		list.add("333");
		//传统
		Iterator<String> it=list.iterator();
		while(it.hasNext()){
			String value=it.next();
			System.out.println(value);
		}
		//增强 for
		for(String s:list)
			System.out.println(s);
	}
	@Test
	public void test2(){
		Map<Integer,String> map=new HashMap<Integer,String>();
		map.put(1, "aaa");
		map.put(2, "bbb");
		map.put(3, "ccc");
		//传统 keyset entryset(购物车)
		Set <Map.Entry<Integer, String>> set=map.entrySet();
		Iterator <Map.Entry<Integer, String>>it=set.iterator();
		while(it.hasNext()){
			Map.Entry<Integer, String>entry=it.next();
			int key=entry.getKey();
			String value=entry.getValue();
			System.out.println(key+"="+value);
		}
```

	//增强 for(重点)
	for(Map.Entry<Integer, String> entry:map.entrySet()){
	int key=entry.getKey();
	String value=entry.getValue();
	}
	}
	}

10.3 自定义泛形——泛型方法

Java 程序中的普通方法、构造方法和静态方法中都可以使用泛型。方法使用泛形前，必须对泛形进行声明，语法：<T>，T 可以是任意字母，但通常必须要大写。<T>通常需放在方法的返回值声明之前。例如： public static <T> void doxx(T t);

练习：

编写一个泛形方法，实现数组指定位置上元素的交换。

编写一个泛形方法，接收一个任意数组，并颠倒数组中的所有元素。

注意：

只有对象类型才能作为泛型方法的实际参数。

在泛型中可以同时有多个类型，例如：

```
public static <K,V> V getValue(K key) { return map.get(key);}
```

10.4 自定义泛形——泛型类

如果一个类多处都要用到同一个泛型，可以把泛形定义在类上（即类级别的泛型），语法格式如下：

```
public class GenericDao<T> {
    private T field1;
    public void save(T obj){}
    public T getId(int id){}
}
```

注意，静态方法不能使用类定义的泛形，而应单独定义泛形。

package cn.itcast.generic;
//自定义带泛型的方法
public class Demo5 {
public void testa(){
a("aaaa");
}
public <T> void a(T t){
 	
}
public <T,E,K> void b(T t,E e,K k){
 	
}
}
	package cn.itcast.generic;
	//自定义类上的泛型
	public class Demo6 <T>{
	public void testa(){
	//a("aaaa");
	}
	public void a(T t){
	
	}
	public <E,K> void b(T t,E e,K k){
	
	}
	//类上的泛型不能作用于静态方法
	public static <T>void c(T t){
	}
	}

10.5 泛型的高级应用——通配符

定义一个方法，接收一个集合，并打印出集合中的所有元素，如下所示：

```
void print (Collection<String> c) {
        for (String e : c) {
            System.out.println(e);
         }
}
```

问题：该方法只能打印保存了 Object 对象的集合，不能打印其它集合。通配符用于解决此类问题，方法的定义可改写为如下形式：

```
void print (Collection<?> c)   {          //Collection<?>(发音为:"collection of unknown")
for (Object e : c) {
      System.out.println(e);
}
}
```

此种形式下需要注意的是：由于 print 方法 c 参数的类型为 Collection<?>，即表示一种不确定的类型，因此在方法体内不能调用与类型相关的方法，例如 add()方法。

总结：使用?通配符主要用于引用对象，使用了?通配符，就只能调对象与类型无关的方法，不能调用对象与类型有关的方法。

10.6 泛型的高级应用——有限制的通配符

限定通配符的上边界：

正确：Vector<?extends Number> x = new Vector<Integer>();

错误：Vector<?extends Number> x = new Vector<String>();

限定通配符的下边界：

正确：Vector<?super Integer> x = new Vector<Number>();

错误：Vector<?super Integer> x = new Vector<Byte>();

问题：以下代码行不行？

```
public void add(List<? extends String> list){
      list.add("abc");
}
```

11 Annotation(注解) 概述

从 JDK 5.0 开始, Java 增加了对元数据(MetaData) 的支持, 也就是 Annotation(注解)。

什么是 Annotation，以及注解的作用?三个基本的 Annotation:

@Override: 限定重写父类方法, 该注解只能用于方法

@Deprecated: 用于表示某个程序元素(类, 方法等)已过时

@SuppressWarnings: 抑制编译器警告.

Annotation 其实就是代码里的特殊标记, 在 Java 技术里注解的典型应用是：可以通过反射技术去得到类里面的注解，以决定怎么去运行类。

掌握注解技术的要点：

如何定义注解

如何反射注解

11.1 自定义 Annotation

定义新的 Annotation 类型使用 @interface 关键字
声明注解的属性：
Annotation 的属性声明方式：String name();
Annotation 属性默认值声明方式：
String name() default “xxx”;
特殊属性 value：如果注解中有一个名称为 value 的属性，那么使用注解时，可以省略 value=部分，例如：@MyAnnotation(“xxx")。

11.2 JDK 的元 Annotation

元 Annotation 指修饰 Annotation 的 Annotation。JDK 中定义了如下元 Annotation：
@Retention: 只能用于修饰一个 Annotation 定义, 用于指定该 Annotation 可以保留的域, @Rentention 包含一个 RetentionPolicy 类型的成员变量, 通过这个变量指定域。
RetentionPolicy.CLASS: 编译器将把注释记录在 class 文件中. 当运行 Java 程序时, JVM 不会保留注释. 这是默认值
RetentionPolicy.RUNTIME:编译器将把注释记录在 class 文件中. 当运行 Java 程序时, JVM 会保留注释. 程序可以通过反射获取该注释
RetentionPolicy.SOURCE: 编译器直接丢弃这种策略的注释
@Target: 指定注解用于修饰类的哪个成员. @Target 包含了一个名为 value 的成员变量.
@Documented: 用于指定被该元 Annotation 修饰的 Annotation 类将被 javadoc 工具提取成文档.
@Inherited: 被它修饰的 Annotation 将具有继承性.如果某个类使用了被 @Inherited 修饰的 Annotation, 则其子类将自动具有该注解

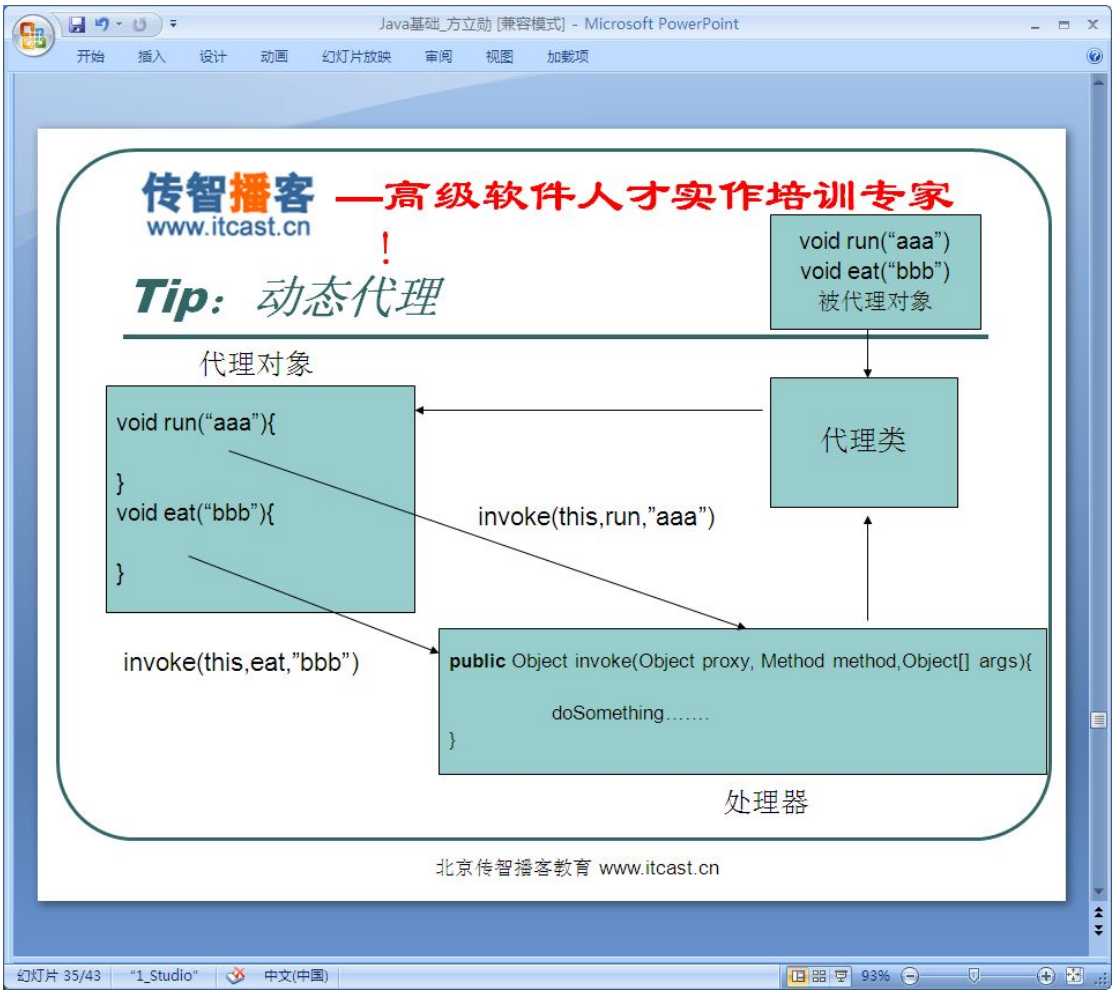
11.3 提取 Annotation 信息

JDK 5.0 在 java.lang.reflect 包下新增了 AnnotationElement 接口, 该接口代表程序中可以接受注释的程序元素
当一个 Annotation 类型被定义为运行时 Annotation 后, 该注释才是运行时可见, 当 class 文件被载入时保存在 class 文件中的 Annotation 才会被虚拟机读取
程序可以调用 AnnotationElement 对象的如下方法来访问 Annotation 信息

11.4 Tip：动态代理

在 java 里，每个对象都有一个类与之对应。
现在要生成某一个对象的代理对象，这个代理对象也要通过一个类来生成，所以首先要编写用于生成代理对象的类。
如何编写生成代理对象的类，两个要素：
代理谁
如何生成代理对象
代理谁？
设计一个类变量，以及一个构造函数，记住代理类代理哪个对象。
如何生成代理对象？
设计一个方法生成代理对象（在方法内编写代码生成代理对象是此处编程的难点）
Java 提供了一个 Proxy 类，调用它的 newInstance 方法可以生成某个对象的代理对象，使用该方法生成代理对象时，需要三个参数：

1.生成代理对象使用哪个类装载器
2.生成哪个对象的代理对象，通过接口指定
3.生成的代理对象的方法里干什么事，由开发人员编写 handler 接口的实现来指定。
初学者必须理解，或不理解必须记住的 2 件事情：
Proxy 类负责创建代理对象时，如果指定了 handler（处理器），那么不管用户调用代理对象的什么方法，该方法都是调用处理器的 invoke 方法。
由于 invoke 方法被调用需要三个参数：代理对象、方法、方法的参数，因此不管代理对象哪个方法调用处理器的 invoke 方法，都必须把自己所在的对象、自己（调用 invoke 方法的方法）、方法的参数传递进来。

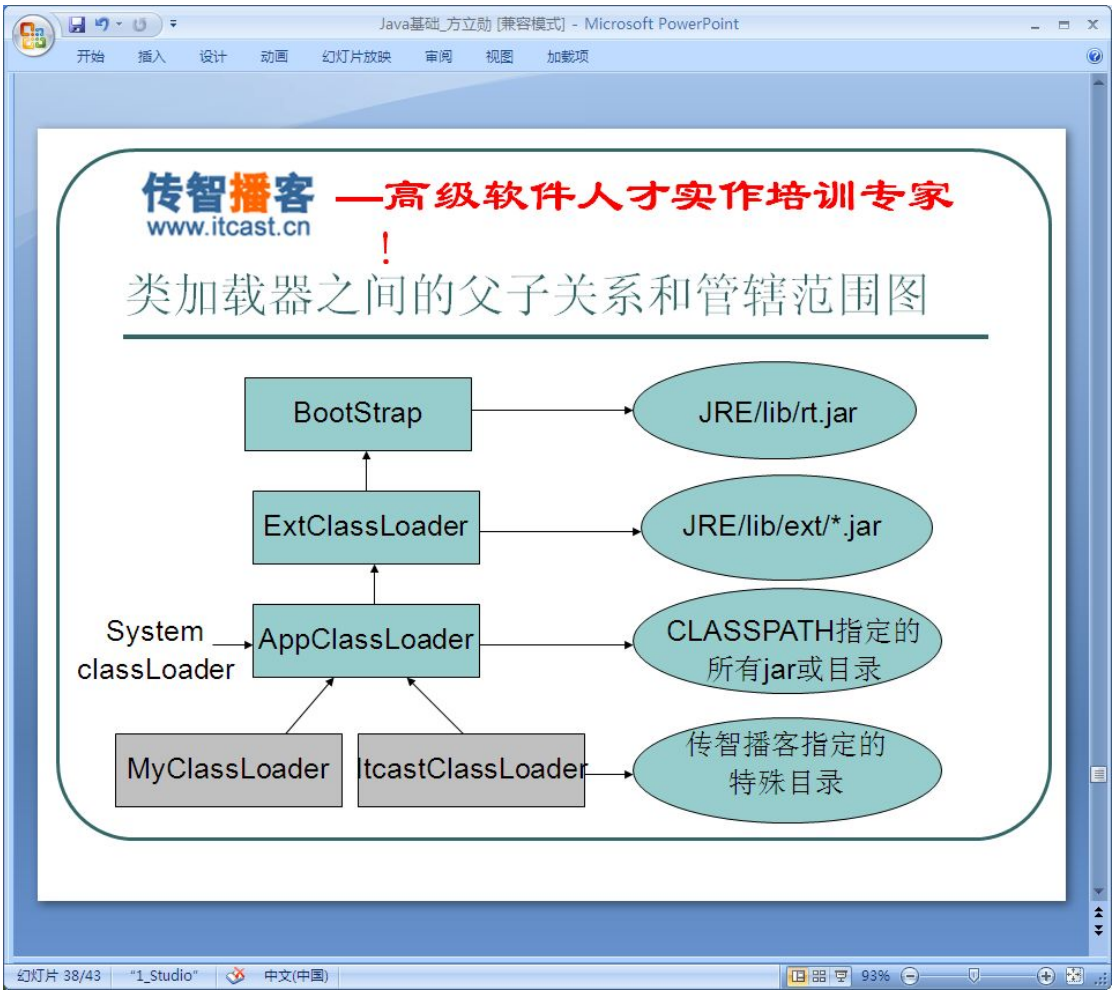


11.5 Tip：动态代理应用

在动态代理技术里，由于不管用户调用代理对象的什么方法，都是调用开发人员编写的处理器的 invoke 方法（这相当于 invoke 方法拦截到了代理对象的方法调用）。
并且，开发人员通过 invoke 方法的参数，还可以在拦截的同时，知道用户调用的是什么方法，因此利用这两个特性，就可以实现一些特殊需求，例如：拦截用户的访问请求，以检查用户是否有访问权限、动态为某个对象添加额外的功能。

11.6 类加载器

类加载器负责将 .class 文件(可能在磁盘上，也可能在网络上) 加载到内存中，并为之生成对应的 java.lang.Class 对象
当 JVM 启动时，会形成由三个类加载器组成的初始类加载器层次结构：



11.7 bootstrap classloader

bootstrap classloader：引导（也称为原始）类加载器，它负责加载 Java 的核心类。这个加载器的是非常特殊的，它实际上不是 java.lang.ClassLoader 的子类，而是由 JVM 自身实现的。可以通过执行以下代码来获得 bootstrap classloader 加载了那些核心类库：

```
URL[] urls=sun.misc.Launcher.getBootstrapClassPath().getURLs();
for (int i = 0; i < urls.length; i++) {
    System.out.println(urls[i].toExternalForm());
}
```

因为 JVM 在启动的时候就自动加载它们,所以不需要在系统属性 CLASSPATH 中指定这些类库

11.8 extension classloader

extension classloader －扩展类加载器，它负责加载 JRE 的扩展目录（JAVA_HOME/jre/lib/ext 或者由 java.ext.dirs 系统属性指定的）中的 JAR 包。这为引入除 Java 核心类以外的新功能提供了一个标准机制。因为默认的扩展目录对所有从同一个 JRE 中启动的 JVM 都是通用的，所以放入这个目录的 JAR 类包对所有的 JVM 和 system classloader 都是可见的。

11.9 system classloader

system classloader －系统（也称为应用）类加载器，它负责在 JVM 被启动时，加载来自在命令 java 中的-classpath 或者 java.class.path 系统属性或者 CLASSPATH 操作系统属性所指定的 JAR 类包和类路径。

可以通过静态方法 ClassLoader.getSystemClassLoader()找到该类加载器。如果没有特别指定，则用户自定义的任何类加载器都将该类加载器作为它的父加载器。

11.10 全盘负责委托机制

classloader 加载类用的是全盘负责委托机制。
全盘负责：即是当一个 classloader 加载一个 Class 的时候，这个 Class 所依赖的和引用的其它 Class 通常也由这个 classloader 负责载入。
委托机制：先让 parent（父）类加载器寻找，只有在 parent 找不到的时候才从自己的类路径中去寻找。
类加载还采用了 cache 机制：如果 cache 中保存了这个 Class 就直接返回它，如果没有才从文件中读取和转换成 Class，并存入 cache，这就是为什么修改了 Class 但是必须重新启动 JVM 才能生效,并且类只加载一次的原因。
HttpUrlConnection
不用 myeclipse 怎么去开发 web 工程

11.11 Tip： DTD 的语法细节：元素定义 1

在 DTD 文档中使用 ELEMENT 声明一个 XML 元素，语法格式如下所示：
<!ELEMENT 元素名称 元素类型>
元素类型可以是元素内容、或类型
如为元素内容：则需要使用()括起来，如
<!ELEMENT 书架 (书名，作者，售价)>
<!ELEMENT 书名 (#PCDATA)>
如为元素类型，则直接书写，DTD 规范定义了如下几种类型：
EMPTY：用于定义空元素，例如
 <hr/>
ANY：表示元素内容为任意类型。
元素内容中可以使用如下方式，描述内容的组成关系
元素内容使用空白符分隔，表示出现顺序没有要求：<!ELEMENT MYFILE (TITLE AUTHOR EMAIL)> ×
用逗号分隔，表示内容的出现顺序必须与声明时一致。<!ELEMENT MYFILE (TITLE,AUTHOR,EMAIL)>
用|分隔，表示任选其一，即多个只能出现一个
<!ELEMENT MYFILE (TITLE|AUTHOR|EMAIL)>
在元素内容中也可以使用+、*、?等符号表示元素出现的次数：
+: 一次或多次 (书+)
?: 0 次或一次 (书?)
: 0 次或多次 (书)
也可使用圆括号( )批量设置，例
<!ELEMENT MYFILE ((TITLE*, AUTHOR?, EMAIL)* | COMMENT)>

11.12 Tip：属性定义

xml 文档中的标签属性需通过 ATTLIST 为其设置属性
语法格式：
<!ATTLIST 元素名
属性名 1 属性值类型 设置说明
属性名 2 属性值类型 设置说明
……
>
属性声明举例：
<!ATTLIST 商品
类别 CDATA #REQUIRED
颜色 CDATA #IMPLIED

>
对应 XML 文件：

```
<商品 类别="服装" 颜色="黄色">…</商品>
<商品 类别="服装">…</商品>
```

设置说明：

#REQUIRED：必须设置该属性

#IMPLIED：可以设置也可以不设置

#FIXED：说明该属性的取值固定为一个值，在 XML 文件中不能为该属性设置其它值。但需要为该属性提供这个值

直接使用默认值：在 XML 中可以设置该值也可以不设置该属性值。若没设置则使用默认值。

举例：

```
<!ATTLIST 页面作者
    姓名 CDATA #IMPLIED
    年龄 CDATA #IMPLIED
    联系信息 CDATA #REQUIRED
    网站职务 CDATA #FIXED "页面作者"
    个人爱好 CDATA "上网"
 >
```

11.13 Tip：常用属性值类型

CDATA：表示属性值为普通文本字符串。

ENUMERATED

ID

ENTITY(实体)

11.14 Tip：属性值类型 ENUMERATED

属性的类型可以是一组取值的列表，在 XML 文件中设置的属性值只能是这个列表中的某个值(枚举)

```
<?xml version = "1.0" encoding="GB2312" standalone="yes"?>
<!DOCTYPE 购物篮 [
    <!ELEMENT 肉 EMPTY>
    <!ATTLIST 肉 品种 ( 鸡肉 | 牛肉 | 猪肉 | 鱼肉 ) "鸡肉">
]>
<购物篮>
    <肉 品种="鱼肉"/>
    <肉 品种="牛肉"/>
    <肉/>
</购物篮>
```

11.15 Tip：属性值类型 ID

表示属性的设置值为一个唯一值。

ID 属性的值只能由字母，下划线开始，不能出现空白字符

11.16 Tip：实体定义

实体用于为一段内容创建一个别名，以后在 XML 文档中就可以使用别名引用这段内容了。

在 DTD 定义中，一条<!ENTITY …>语句用于定义一个实体。

实体可分为两种类型：引用实体和参数实体。

11.17 Tip：实体定义 引用实体

引用实体主要在 XML 文档中被应用
语法格式：
<!ENTITY 实体名称 “实体内容” >：直接转变成实体内容
引用方式：
&实体名称;
举例：

```
<!ENTITY copyright “I am a programmer">
……
&copyright;
```

11.18 Tip：实体定义 参数实体

参数实体被 DTD 文件自身使用
语法格式：

```
<!ENTITY % 实体名称 "实体内容" >
```

引用方式：
%实体名称;
举例 1：

```
<!ENTITY % TAG_NAMES "姓名 | EMAIL | 电话 | 地址">

<!ELEMENT 个人信息 (%TAG_NAMES; | 生日)>
<!ELEMENT 客户信息 (%TAG_NAMES; | 公司名)>
```

举例 2：

```
<!ENTITY % common.attributes
" id ID #IMPLIED
account CDATA #REQUIRED "
>
...
<!ATTLIST purchaseOrder %common.attributes;>
<!ATTLIST item %common.attributes;>
```

11.19 Tip：XML 解析技术概述

XML 解析方式分为两种：dom 和 sax
dom：(Document Object Model, 即文档对象模型) 是 W3C 组织推荐的处理 XML 的一种方式。
sax： (Simple API for XML) 不是官方标准，但它是 XML 社区事实上的标准，几乎所有的 XML 解析器都支持它。
XML 解析器
Crimson、Xerces 、Aelfred2
XML 解析开发包
Jaxp、Jdom、dom4j

11.20 Tip：JAXP

JAXP 开发包是 J2SE 的一部分，它由 javax.xml、org.w3c.dom 、org.xml.sax 包及其子包组成
在 javax.xml.parsers 包中，定义了几个工厂类，程序员调用这些工厂类，可以得到对 xml 文档进行解析的 DOM 或

SAX 的解析器对象。

11.21 Tip：使用 JAXP 进行 DOM 解析

javax.xml.parsers 包中的 DocumentBuilderFactory 用于创建 DOM 模式的解析器对象 ， DocumentBuilderFactory 是一个抽象工厂类，它不能直接实例化，但该类提供了一个 newInstance 方法 ，这个方法会根据本地平台默认安装的解析器，自动创建一个工厂的对象并返回。

11.22 Tip：获得 JAXP 中的 DOM 解析器

调用 DocumentBuilderFactory.newInstance() 方法得到创建 DOM 解析器的工厂。

调用工厂对象的 newDocumentBuilder 方法得到 DOM 解析器对象。

调用 DOM 解析器对象的 parse() 方法解析 XML 文档，得到代表整个文档的 Document 对象，进行可以利用 DOM 特性对整个 XML 文档进行操作了。

```
package cn.itcast.xml;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.junit.Test;
import org.w3c.dom.Document;

public class Demo2 {
	public static void main(String[] args) throws Exception {
		// TODO Auto-generated method stub
		//1.创建工厂
		  DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
		//2。得到 dom 解析器
		  DocumentBuilder builder=factory.newDocumentBuilder();
		//3。解析 xml 文档，得到代表文档的 document
		  Document document=builder.parse("src/book.xml");
	}
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<书架>
	<书>
		<书名  name="dddd">java web 就业</书名>
		<作者>张孝祥</作者>
		<售价>40</售价>
	</书>
	<书>
		<书名>C++教程</书名>
		<作者>自己</作者>
		<售价>50</售价>
	</书>
</书架>
```

```
package cn.itcast.xml;

import java.io.File;
import java.io.FileOutputStream;
import java.io.OutputStream;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.junit.Test;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

//使用 dom 方式对 xml 文档进行 CRUD
public class Demo3 {
	//读取<书名>C++教程</书名>
	public void read1() throws Exception
	{
		DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
		DocumentBuilder builder=factory.newDocumentBuilder();
		Document document=builder.parse("src/book.xml");
		NodeList list=document.getElementsByTagName("书名");
		Node node=list.item(1);
		String content=node.getTextContent();
		System.out.println(content);//C++教程
	}
	//得到文档中所有标签
	public void read2() throws Exception
	{
		DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
		DocumentBuilder builder=factory.newDocumentBuilder();
		Document document=builder.parse("src/book.xml");
		NodeList list=document.getElementsByTagName("书名");
		Node root=document.getElementsByTagName("书架").item(0);
		list(root);
	}
	private void list(Node node) {
		Node child;
		if (node instanceof Element)
			System.out.println(node.getNodeName());
```

```
        NodeList nodelist=node.getChildNodes();
        for (int i=0;i<nodelist.getLength();i++)
        {
            child=nodelist.item(i);
            list(child);
        }
    }
    //得到文档中标签属性<书名  name="xxxx">java web 就业</书名>
    public void read3() throws Exception
    {
        DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
        DocumentBuilder builder=factory.newDocumentBuilder();
        Document document=builder.parse("src/book.xml");


        NodeList list=document.getElementsByTagName("书名");
        Node node=list.item(0);
        if(node.hasAttributes()){
            NamedNodeMap nodemap=node.getAttributes();
            for(int i=0;i<nodemap.getLength();i++)
            {
                Node nd=nodemap.item(i);
                String strname=nd.getNodeName();
                String strval=nd.getNodeValue();
                System.out.println(strname+":"+strval);//name:dddd

            }
        }
        Element node2=(Element)list.item(0);
        String str3=node2.getAttribute("name");
        System.out.println("__"+str3);//__dddd
    }
    //<售价>10</售价>
    public void add() throws Exception{
        DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
        DocumentBuilder builder=factory.newDocumentBuilder();
        Document document=builder.parse("src/book.xml");
        //创建节点
        Element price=document.createElement("售价");
        price.setTextContent("59.0 元");
        //把创建的节点放到第一本书上
        document.getElementsByTagName("书").item(0).appendChild(price);
        //把跟新后的内容写回文档
        Transformer transformer=TransformerFactory.newInstance().newTransformer();
        DOMSource source=new DOMSource(document);
        FileOutputStream outstream =new FileOutputStream(new File("src/outbook.xml"));
        StreamResult reslut=new StreamResult(outstream);
```

```
        transformer.transform(source, reslut);
        outstream.close();
    }

    //向文档中指定位置上添加节点   <售价>10</售价>
    public void add2() throws Exception{
        DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
        DocumentBuilder builder=factory.newDocumentBuilder();
        Document document=builder.parse("src/book.xml");
        //创建节点
        Element price=document.createElement("售价");
        price.setTextContent("59.0 元");
        //得到参考节点
        Element refNode=(Element)document.getElementsByTagName("售价").item(0);
        //得到挂崽的节点
        Element book=(Element)document.getElementsByTagName("书").item(0);
        //把创建的节点放到第一本书上
        document.getElementsByTagName("书").item(0).appendChild(price);
        //  往 book 节点指定位置插值
        book.insertBefore(price, refNode);
        Transformer transformer=TransformerFactory.newInstance().newTransformer();
        DOMSource source=new DOMSource(document);
        FileOutputStream outstream =new FileOutputStream(new File("src/outbook2.xml"));
        StreamResult reslut=new StreamResult(outstream);
        transformer.transform(source, reslut);
        outstream.close();
    }
    //向文档节点 添加属性 <售价>10</售价>
    public void addAtt() throws Exception{
        DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
        DocumentBuilder builder=factory.newDocumentBuilder();
        Document document=builder.parse("src/book.xml");

        //得到参考节点
        Element refNode=(Element)document.getElementsByTagName("售价").item(0);
        refNode.setAttribute("addAtrr","new cha ru value");

        Transformer transformer=TransformerFactory.newInstance().newTransformer();
        DOMSource source=new DOMSource(document);
        FileOutputStream outstream =new FileOutputStream(new File("src/outbook3.xml"));
        StreamResult reslut=new StreamResult(outstream);
        transformer.transform(source, reslut);
        outstream.close();
    }
    //删除 <售价>10</售价>
    public void delete() throws Exception{
        DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
```

```
        DocumentBuilder builder=factory.newDocumentBuilder();
        Document document=builder.parse("src/book.xml");

        //得到要删除的节点
        Element refNode=(Element)document.getElementsByTagName("售价").item(0);
        refNode.getParentNode().removeChild(refNode);

        Transformer transformer=TransformerFactory.newInstance().newTransformer();
        DOMSource source=new DOMSource(document);
        FileOutputStream outstream =new FileOutputStream(new File("src/outbook3.xml"));
        StreamResult reslut=new StreamResult(outstream);
        transformer.transform(source, reslut);
        outstream.close();
    }
    //更新  售价
    public void update() throws Exception{
        DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
        DocumentBuilder builder=factory.newDocumentBuilder();
        Document document=builder.parse("src/book.xml");

        //得到要更新的节点
        Element refNode=(Element)document.getElementsByTagName("售价").item(0);
        refNode.setTextContent("1000");

        Transformer transformer=TransformerFactory.newInstance().newTransformer();
        DOMSource source=new DOMSource(document);
        FileOutputStream outstream =new FileOutputStream(new File("src/outbook3.xml"));
        StreamResult reslut=new StreamResult(outstream);
        transformer.transform(source, reslut);
        outstream.close();
    }
}
```

11.23 调虚拟机内存大小

-Xmx83886080 C:\Program Files\Java\jdk1.7.0\docs\technotes\tools\windows\java.html
-Xmx81920k
-Xmx80m

11.24 Tip：DOM 编程

DOM 模型(document object model)
DOM 解析器在解析 XML 文档时，会把文档中的所有元素，按照其出现的层次关系，解析成一个个 Node 对象(节点)。
在 dom 中，节点之间关系如下：
位于一个节点之上的节点是该节点的父节点(parent)
一个节点之下的节点是该节点的子节点（children）

同一层次，具有相同父节点的节点是兄弟节点（sibling）
一个节点的下一个层次的节点集合是节点后代(descendant)
父、祖父节点及所有位于节点上面的，都是节点的祖先(ancestor)
节点类型（下页 ppt）
Node 对象提供了一系列常量来代表结点的类型，当开发人员获得某个 Node 类型后，就可以把 Node 节点转换成相应的节点对象(Node 的子类对象)，以便于调用其特有的方法。（查看 API 文档）
Node 对象提供了相应的方法去获得它的父结点或子结点。编程人员通过这些方法就可以读取整个 XML 文档的内容、或添加、修改、删除 XML 文档的内容了。

11.25 Tip：DOM 方式解析 XML 文件

DOM 解析编程
遍历所有节点
查找某一个节点
删除结点
更新结点
添加节点

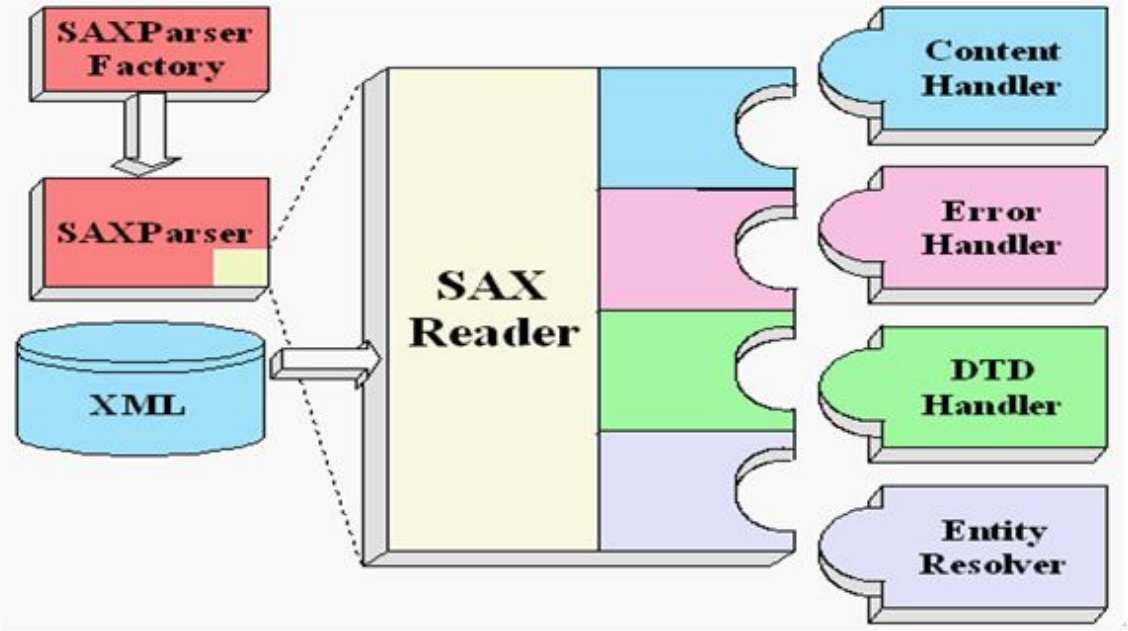
11.26 Tip：更新 XML 文档

javax.xml.transform 包中的 Transformer 类用于把代表 XML 文件的 Document 对象转换为某种格式后进行输出，例如把 xml 文件应用样式表后转成一个 html 文档。利用这个对象，当然也可以把 Document 对象又重新写入到一个 XML 文件中。
Transformer 类通过 transform 方法完成转换操作，该方法接收一个源和一个目的地。我们可以通过：
javax.xml.transform.dom.DOMSource 类来关联要转换的 document 对象，
用 javax.xml.transform.stream.StreamResult 对象来表示数据的目的地。
Transformer 对象通过 TransformerFactory 获得。

11.27 Tip：SAX 解析

在使用 DOM 解析 XML 文档时，需要读取整个 XML 文档，在内存中构架代表整个 DOM 树的 Doucment 对象，从而再对 XML 文档进行操作。此种情况下，如果 XML 文档特别大，就会消耗计算机的大量内存，并且容易导致内存溢出。

SAX 解析允许在读取文档的时候，即对文档进行处理，而不必等到整个文档装载完才会文档进行操作。
SAX 采用事件处理的方式解析 XML 文件，利用 SAX 解析 XML 文档，涉及两个部分：解析器和事件处理器：
解析器可以使用 JAXP 的 API 创建，创建出 SAX 解析器后，就可以指定解析器去解析某个 XML 文档。
解析器采用 SAX 方式在解析某个 XML 文档时，它只要解析到 XML 文档的一个组成部分，都会去调用事件处理器的一个方法，解析器在调用事件处理器的方法时，会把当前解析到的 xml 文件内容作为方法的参数传递给事件处理器。
事件处理器由程序员编写，程序员通过事件处理器中方法的参数，就可以很轻松地得到 sax 解析器解析到的数据，从而可以决定如何对数据进行处理。



阅读 ContentHandler API 文档，常用方法：startElement、endElement、characters

11.28 Tip：SAX 方式解析 XML 文档

使用 SAXParserFactory 创建 SAX 解析工厂

```
SAXParserFactory spf = SAXParserFactory.newInstance();
```

通过 SAX 解析工厂得到解析器对象

```
SAXParser sp = spf.newSAXParser();
```

通过解析器对象得到一个 XML 的读取器

```
XMLReader xmlReader = sp.getXMLReader();
```

设置读取器的事件处理器

```
xmlReader.setContentHandler(new BookParserHandler());
```

解析 xml 文件

```
xmlReader.parse("book.xml");
```

```
package cn.itcast.sax;

import java.io.IOException;
import javax.xml.parsers.*;
import org.xml.sax.*;
public class Demo1 {
    /**
     * sax 解析 xml 文档
     */
    public static void main(String[] args) throws Exception {
        //1.创建解析工厂
        SAXParserFactory factory=SAXParserFactory.newInstance();
        //2.得到解析器
        SAXParser sp=factory.newSAXParser();
        //3.得到读取器
        XMLReader reader=sp.getXMLReader();
```

```
            //4.设置内容处理器
            reader.setContentHandler(new ListHandler());
            //5.读取 xml 文档内容
            reader.parse("src/book.xml");
        }
}
//得到 xml 文档所有内容
class ListHandler implements ContentHandler{

        @Override
        public void characters(char[] ch, int start, int length)
                    throws SAXException {
            System.out.println(new String(ch,start,length));
        }
        @Override
        public void endDocument() throws SAXException {
            // TODO Auto-generated method stub

        }
        @Override
        public void endElement(String uri, String localName, String qName)
                    throws SAXException {
            System.out.println("</"+qName+">");
        }
        @Override
        public void endPrefixMapping(String prefix) throws SAXException {
                    }
        @Override
        public void ignorableWhitespace(char[] ch, int start, int length)
                    throws SAXException {

        }
        @Override
        public void processingInstruction(String target, String data)
                    throws SAXException {

        }
        @Override
        public void setDocumentLocator(Locator locator) {

        }
        @Override
        public void skippedEntity(String name) throws SAXException {

        }
        @Override
        public void startDocument() throws SAXException {
```

```
	}
	@Override
	public void startElement(String uri, String localName, String qName,
			Attributes atts) throws SAXException {
		System.out.println("<"+qName+">");
		for (int i=0;atts!=null && i<atts.getLength();i++){
			String attName=atts.getQName(i);
			String attValue=atts.getValue(i);
			System.out.println(attName+"="+attValue);
		}
	}
	@Override
	public void startPrefixMapping(String prefix, String uri)
			throws SAXException {

	}
}
```

```
package cn.itcast.sax;
import java.io.IOException;
import javax.xml.parsers.*;
import org.xml.sax.*;

public class Demo2 {
	/**
	 * sax 解析 xml 文档
	 */
	public static void main(String[] args) throws Exception {
		//1.创建解析工厂
		SAXParserFactory factory=SAXParserFactory.newInstance();
		//2.得到解析器
		SAXParser sp=factory.newSAXParser();
		//3.得到读取器
		XMLReader reader=sp.getXMLReader();
		//4.设置内容处理器
		reader.setContentHandler(new TagValueHandler());
		//5.读取 xml 文档内容
		reader.parse("src/book.xml");
	}
}
//获取指定标签  作者 的值
class TagValueHandler extends DefaultHandler{
	private String currentTag;//记住当前解析器得到的是什么标签
	private int needNumber=2;//记住想获取第几个作者标签的值
	private int currentNumber;//当前解析的是第几个
	@Override
```

```
    public void startElement(String uri, String localName, String qName,
            Attributes attributes) throws SAXException {
        currentTag=qName;
        if("作者".equals(currentTag))
            currentNumber++;
    }
    @Override
    public void characters(char[] ch, int start, int length)    throws SAXException {
        if("作者".equals(currentTag)&& currentNumber==needNumber){
            System.out.println(new String(ch,start,length));
        }
    }
    @Override
    public void endElement(String uri, String localName, String qName)
            throws SAXException {
        currentTag=null;
    }
}
```

```
package cn.itcast.sax;
import java.io.IOException;
import java.util.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
public class Demo3 {

    /**
     * sax 解析 xml 文档
     */
    public static void main(String[] args) throws Exception {
        //1.创建解析工厂
        SAXParserFactory factory=SAXParserFactory.newInstance();
        //2.得到解析器
        SAXParser sp=factory.newSAXParser();
        //3.得到读取器
        XMLReader reader=sp.getXMLReader();
        //4.设置内容处理器
        BeanListHandler handle=new BeanListHandler();
        reader.setContentHandler(handle);
        //5.读取 xml 文档内容
        reader.parse("src/book.xml");

        List<Book> list=handle.getList();
        list.iterator();

    }
}
```

```
//把每一本书封装到一个 book 对象，并把 book 对象
class BeanListHandler extends DefaultHandler{
	private List<Book> list=new ArrayList<Book>();
	public List<Book> getList() {
		return list;
	}

	private String currentTag;
	private Book book;
	@Override
	public void startElement(String uri, String localName, String qName,
			Attributes attributes) throws SAXException {
		currentTag=qName;
		if("书".equals(currentTag)){
			book=new Book();
		}
	}
	@Override
	public void characters(char[] ch, int start, int length)		throws SAXException {
		if("书名".equals(currentTag)){
			book.setName(new String(ch,start,length));
		}
		if("作者".equals(currentTag)){
			book.setAuthor(new String(ch,start,length));
		}
		if("售价".equals(currentTag)){
			book.setPrice(new String(ch,start,length));
		}
	}

	@Override
	public void endElement(String uri, String localName, String qName)	throws SAXException {
		if(qName.equals("书")){
			list.add(book);
			book=null;
		}
		currentTag=null; //这句有必要
	}
}
```

11.29 Tip：DOM4J 解析 XML 文档

Dom4j 是一个简单、灵活的开放源代码的库。Dom4j 是由早期开发 JDOM 的人分离出来而后独立开发的。与 JDOM 不同的是，dom4j 使用接口和抽象基类，虽然 Dom4j 的 API 相对要复杂一些，但它提供了比 JDOM 更好的灵活性。Dom4j 是一个非常优秀的 Java XML API，具有性能优异、功能强大和极易使用的特点。现在很多软件采用的 Dom4j，例如 Hibernate，包括 sun 公司自己的 JAXM 也用了 Dom4j。

使用 Dom4j 开发，需下载 dom4j 相应的 jar 文件。

```
    //读第 2 本书的信息 <书名  name="xxxx">C++教程</书名>
    private static void read() throws DocumentException {
        SAXReader reader=new SAXReader();
        Document document=reader.read(new File("src/book.xml"));
        Element root=document.getRootElement();
        Element book=(Element) root.elements("书").get(1);
        String value=book.element("书名").getText();
        String value2=book.element("书名").attributeValue("name");
        System.out.println(value  +";"+value2);
    }
```

```
//在第一本上添加一个新的售价
public void add() throws Exception{
    SAXReader reader=new SAXReader();
    Document document=reader.read(new File("src/book.xml"));
    Element book=document.getRootElement().element("书");
    book.addElement("售价").setText("111");
    //XMLWriter writer=new XMLWriter(new FileWriter("src/book.xml"));
    //XMLWriter writer=new XMLWriter(new OutputStreamWriter(new
FileOutputStream("src/book.xml"), "UTF-8"));

    OutputFormat format = OutputFormat.createPrettyPrint();
    format.setEncoding("gb2312");
    XMLWriter writer=new XMLWriter(new OutputStreamWriter(new FileOutputStream("src/book.xml"),
"gb2312"),format);

    writer.write(document);
    writer.close();
}
```

```
//在第一本书指定位置上添加一个新的售价,更改 List 集合
public void add2() throws Exception{
    SAXReader reader=new SAXReader();
    Document document=reader.read(new File("src/book.xml"));

    Element book=document.getRootElement().element("书");
    List list=book.elements();//[书名，作者，售价]

    Element price=DocumentHelper.createElement("售价");
    price.setText("309 元");
    list.add(2,price);

    OutputFormat format = OutputFormat.createPrettyPrint();
    format.setEncoding("gb2312");
    XMLWriter writer=new XMLWriter(new OutputStreamWriter(new FileOutputStream("src/book.xml"),
"gb2312"),format);
```

```
        writer.write(document);
        writer.close();
}
```

```
//删除上面的节点
public void delete() throws Exception {
        SAXReader reader=new SAXReader();
        Document document=reader.read(new File("src/book.xml"));

        Element price=document.getRootElement().element("书").element("售价");
        price.getParent().remove(price);
}
```

```
//更新节点
SAXReader reader=new SAXReader();
Document document=reader.read(new File("src/book.xml"));

Element book=(Element) document.getRootElement().elements("书").get(1);
book.element("作者").setText("张三");
OutputFormat format = OutputFormat.createPrettyPrint();
format.setEncoding("gb2312");
XMLWriter writer=new XMLWriter(new OutputStreamWriter(new FileOutputStream("src/book.xml"),
"gb2312"),format);

writer.write(document);
writer.close();
```

```
//应用 xpath 提取 xml 文档的数据,需要包 jaxen-1.1-beta-6.jar
SAXReader reader=new SAXReader();
Document document=reader.read(new File("src/book.xml"));
String value=document.selectSingleNode("//作者").getText();//第一个值
System.out.println(value);
```

11.30 Tip：Document 对象

DOM4j 中，获得 Document 对象的方式有三种：

1.读取 XML 文件,获得 document 对象

```
SAXReader reader = new SAXReader();
Document    document = reader.read(new File("input.xml"));
```

2.解析 XML 形式的文本,得到 document 对象.

```
String text = "<members></members>";
Document document = DocumentHelper.parseText(text);
```

3.主动创建 document 对象.

```
Document document = DocumentHelper.createDocument();
//创建根节点
Element root = document.addElement("members");
```

11.31 Tip：节点对象

1.获取文档的根节点.

```
        Element root = document.getRootElement();
```
2.取得某个节点的子节点.
```
      Element element=node.element(“书名");
```
3.取得节点的文字
```
        String text=node.getText();
```
4.取得某节点下所有名为“member”的子节点，并进行遍历.
```
 List nodes = rootElm.elements("member");
  for (Iterator it = nodes.iterator(); it.hasNext();) {
    Element elm = (Element) it.next();
   // do something
 }
```

5.对某节点下的所有子节点进行遍历.
```
    for(Iterator it=root.elementIterator();it.hasNext();){
          Element element = (Element) it.next();
          // do something
   }
```

6.在某节点下添加子节点.
```
Element ageElm = newMemberElm.addElement("age");
```
7.设置节点文字.
```
 element.setText("29");
```

8.删除某节点.
```
//childElm 是待删除的节点,parentElm 是其父节点
      parentElm.remove(childElm);
```
9.添加一个 CDATA 节点.
```
Element contentElm = infoElm.addElement("content");
contentElm.addCDATA(diary.getContent());
```

11.32 Tip：节点对象属性

1.取得某节点下的某属性
```
    Element root=document.getRootElement();
    //属性名 name
                Attribute attribute=root.attribute("size");
```
2.取得属性的文字
```
  String text=attribute.getText();
```
3.删除某属性
```
 Attribute attribute=root.attribute("size");
 root.remove(attribute);
```
3.遍历某节点的所有属性
```
   Element root=document.getRootElement();
  for(Iterator it=root.attributeIterator();it.hasNext();){
       Attribute attribute = (Attribute) it.next();
       String text=attribute.getText();
       System.out.println(text);
```

```
}
```

4.设置某节点的属性和文字.

```
newMemberElm.addAttribute("name", "sitinspring");
```

5.设置属性的文字

```
Attribute attribute=root.attribute("name");
attribute.setText("sitinspring");
```

11.33 Tip：将文档写入 XML 文件

1.文档中全为英文,不设置编码,直接写入的形式.

```
XMLWriter writer = new XMLWriter(new    FileWriter("output.xml"));
writer.write(document);
writer.close();
```

2.文档中含有中文,设置编码格式写入的形式.

```
OutputFormat format = OutputFormat.createPrettyPrint();
//  指定 XML 编码
        format.setEncoding("GBK");
XMLWriter writer = new XMLWriter(newFileWriter("output.xml"),format);
writer.write(document);
writer.close();
```

11.34 Tip：Dom4j 在指定位置插入节点

1.得到插入位置的节点列表（list）

2.调用 list.add(index,elemnent)，由 index 决定 element 的插入位置。

Element 元素可以通过 DocumentHelper 对象得到。示例代码：

```
Element aaa = DocumentHelper.createElement("aaa");
aaa.setText("aaa");

List list = root.element("书").elements();
list.add(1, aaa);

//更新 document
```

11.35 Tip：字符串与 XML 的转换

1.将字符串转化为 XML

```
        String text = "<members> <member>sitinspring</member></members>";
Document document = DocumentHelper.parseText(text);
```

2.将文档或节点的 XML 转化为字符串.

```
        SAXReader reader = new SAXReader();
Document    document = reader.read(new File("input.xml"));
Element root=document.getRootElement();
```

```
String docXmlText=document.asXML();

String rootXmlText=root.asXML();
Element memberElm=root.element("member");
String memberXmlText=memberElm.asXML();
```

11.36 XML Schema

XML Schema 也是一种用于定义和描述 XML 文档结构与内容的模式语言，其出现是为了克服 DTD 的局限性
XML Schema VS DTD：
XML Schema 符合 XML 语法结构。
DOM、SAX 等 XML API 很容易解析出 XML Schema 文档中的内容。
XML Schema 对名称空间支持得非常好。
XML Schema 比 XML DTD 支持更多的数据类型，并支持用户自定义新的数据类型。
XML Schema 定义约束的能力非常强大，可以对 XML 实例文档作出细致的语义限制。
XML Schema 不能像 DTD 一样定义实体，比 DTD 更复杂，但 Xml Schema 现在已是 w3c 组织的标准，它正逐步取代 DTD。

11.37 Schema 约束快速入门

XML Schema 文件自身就是一个 XML 文件，但它的扩展名通常为.xsd。
一个 XML Schema 文档通常称之为模式文档(约束文档)，遵循这个文档书写的 xml 文件称之为实例文档。
和 XML 文件一样，一个 XML Schema 文档也必须有一个根结点，但这个根结点的名称为 Schema。
编写了一个 XML Schema 约束文档后，通常需要把这个文件中声明的元素绑定到一个ＵＲＩ地址上，在 XML Schema 技术中有一个专业术语来描述这个过程，即把 XML Schema 文档声明的元素绑定到一个名称空间上，以后 XML 文件就可以通过这个 URI（即名称空间）来告诉解析引擎，xml 文档中编写的元素来自哪里，被谁约束。

11.38 Schema 入门案例

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                        targetNamespace="http://www. itcast.cn"
                        elementFormDefault="qualified">
    <xs:element name='书架' >
        <xs:complexType>
            <xs:sequence maxOccurs='unbounded' >
                <xs:element name='书' >
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name='书名' type='xs:string' />
                            <xs:element name='作者' type='xs:string' />
                            <xs:element name='售价' type='xs:string' />
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
```

```
</xs:schema>
<?xml version="1.0" encoding="UTF-8"?>

<itcast:书架 xmlns:itcast="http://www.itcast.cn"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://www.itcast.cn book.xsd">

    <itcast:书>
        <itcast:书名>JavaScript 网页开发</itcast:书名>
        <itcast:作者>张孝祥</itcast:作者>
        <itcast:售价>28.00 元</itcast:售价>
    </itcast:书>
</itcast:书架>
```

11.39 名称空间的概念

在 XML Schema 中，每个约束模式文档都可以被赋以一个唯一的名称空间，名称空间用一个唯一的 URI（Uniform Resource Identifier，统一资源标识符）表示。 在 Xml 文件中书写标签时，可以通过名称空间声明（xmlns），来声明当前编写的标签来自哪个 Schema 约束文档。如：

```
<itcast:书架 xmlns:itcast="http://www.itcast.cn">
    <itcast:书>……</itcast:书>
</itcast:书架>
```

此处使用 itcast 来指向声明的名称，以便于后面对名称空间的引用。

注意：名称空间的名字语法容易让人混淆，尽管以 http:// 开始，那个 URL 并不指向一个包含模式定义的文件。事实上，这个 URL：http://www.itcast.cn 根本没有指向任何文件，只是一个分配的名字。

11.40 使用名称空间引入 Schema

为了在一个 XML 文档中声明它所遵循的 Schema 文件的具体位置，通常需要在 Xml 文档中的根结点中使用 schemaLocation 属性来指定，例如：

```
<itcast:书架 xmlns:itcast="http://www.itcast.cn"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.itcast.cn book.xsd">
```

schemaLocation 此属性有两个值。第一个值是需要使用的命名空间。第二个值是供命名空间使用的 XML schema 的位置，两者之间用空格分隔。

注意，在使用 schemaLocation 属性时，也需要指定该属性来自哪里。

11.41 使用默认名称空间

基本格式：

```
xmlns="URI"
```

举例：

```
<书架 xmlns="http://www.it315.org/xmlbook/schema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.itcast.cn book.xsd">
    <书>
    <书名>JavaScript 网页开发</书名>
    <作者>张孝祥</作者>
    <售价>28.00 元</售价>
```

```
    </书>
  <书架>
```

11.42 使用名称空间引入多个 XML Schema 文档

文件清单：xmlbook.xml

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

<书架 xmlns="http://www.it315.org/xmlbook/schema"
    xmlns:demo="http://www.it315.org/demo/schema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.it315.org/xmlbook/schema http://www.it315.org/xmlbook.xsd
        http://www.it315.org/demo/schema http://www.it315.org/demo.xsd">
    <书>
        <书名>JavaScript 网页开发</书名>
        <作者>张孝祥</作者>
        <售价 demo:币种=”人民币”>28.00 元</售价>
    </书>
</书架>
```

11.43 不使用名称空间引入 XML Schema 文档

文件清单：xmlbook.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<书架 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="xmlbook.xsd">
    <书>
        <书名>JavaScript 网页开发</书名>
        <作者>张孝祥</作者>
        <售价>28.00 元</售价>
    </书>
</书架>
```

11.44 在 XML Schema 文档中声明名称空间

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                targetNamespace="http://www. itcast.cn"
                elementFormDefault="qualified">
<xs:schema>
```

targetNamespace 元素用于指定 schema 文档中声明的元素属于哪个名称空间。
elementFormDefault 元素用于指定，该 schema 文档中声明的根元素及其所有子元素都属于 targetNamespace 所指定的名称空间。

12 HTTP 协议

12.1 什么是 HTTP 协议

客户端连上 web 服务器后，若想获得 web 服务器中的某个 web 资源，需遵守一定的通讯格式，HTTP 协议用于定义客户端与 web 服务器通迅的格式。

使用 telnet 程序连上 web 服务器，并使用 HTTP 协议获取某个页面，以快速了解 HTTP 协议的作用。

安装 IE 浏览器插件 HttpWatch，查看 IE 浏览器通过 HTTP 协议获取某个页面。

```
import java.io.*;
import java.net.*;

public class Server {
	public static void main(String[] args) throws Exception {
		ServerSocket server=new ServerSocket(9999);
		Socket sock=server.accept();
		FileInputStream in=new FileInputStream("d:\\qq\\1.html");
		OutputStream out=sock.getOutputStream();
		int len=0;
		byte buffer[]=new byte[1024];
		while((len=in.read(buffer))>0){
			out.write(buffer,0,len);
		}
		in.close();
		out.close();
		sock.close();
		server.close();
	}
}
```

Weblogic tomcat WebSphere
Fport 工具查看端口占用的程序

Java_home 环境变量的设置问题
只要在 setclasspath.bat 批处理文件第一次使用 JAVA_HOME 环境变量之前的任何地方，将 JAVA_HOME 环境变量设置为 JDK 的主目录，就可以使用 startup.bat 文件启动 Tomcat 了。
端口占用问题
Catalina_home 环境变量的设置问题 指明 Tomcat 在哪里，当硬盘上有多个 Tomcat 时，点任何一个 startup.bat 时，都是运行同一个。（一般不配）

12.2 Tip：配置虚拟目录

Web 应用开发好后，若想供外界访问，需要把 web 应用交给 web 服务器管理，这个过程称之为配置虚拟目录。tomcat 服务器会自动管理 webapps 目录下的所有 web 应用，并把它映射成虚似目录。换句话说，只要把 web 项目放置到 tomcat 服务器的 webapps 目录中，不需要做其它设置，这个 web 应用就可以直接被外界访问了。

对计算机中的任意位置的 WEB 应用，若想被外界访问，就需要手工通知 web 服务器去管理，即通知 web 服务器把其映射成虚似目录，这样才能供外界访问。

<Context path="/itcast" docBase="c:\app" />

12.3 Tip2：HTTP 协议简介

HTTP 是 hypertext transfer protocol（超文本传输协议）的简写，它是 TCP/IP 协议的一个应用层协议，用于定义 WEB 浏览器与 WEB 服务器之间交换数据的过程。

HTTP 协议是学习 JavaWEB 开发的基石，不深入了解 HTTP 协议，就不能说掌握了 WEB 开发，更无法管理和维护一些复杂的 WEB 站点。

HTTP 协议的版本：HTTP/1.0、HTTP/1.1

12.4 Tip3：HTTP1.0 和 HTTP1.1 的区别

在 HTTP1.0 协议中，客户端与 web 服务器建立连接后，只能获得一个 web 资源。

HTTP1.1 协议，允许客户端与 web 服务器建立连接后，在一个连接上获取多个 web 资源。

使用 telnet 举例说明。

一个好多同学搞不清楚的问题：

一个 web 页面中，使用 img 标签引用了三幅图片，当客户端访问服务器中的这个 web 页面时，客户端总共会访问几次服务器，即向服务器发送了几次 HTTP 请求。

12.5 Tip4：HTTP 请求

客户端连上服务器后，向服务器请求某个 web 资源，称之为客户端向服务器发送了一个 HTTP 请求。一个完整的 HTTP 请求包括如下内容：

一个请求行、若干消息头、以及实体内容，其中的一些消息头和实体内容都是可选的，消息头和实体内容之间要用空行隔开。如下所示：

GET /books/java.html HTTP/1.1	请求行
Accept: */*	
Accept-Language: en-us	
Connection: Keep-Alive	多个消息头
Host: localhost	
Referer: http://localhost/links.asp	
User-Agent: Mozilla/4.0	
Accept-Encoding: gzip, deflate	
	一个空行

12.6 Tip5：HTTP 请求的细节——请求行

请求行中的 GET 称之为请求方式，请求方式有：

POST、GET、HEAD、OPTIONS、DELETE、TRACE、PUT

常用的有：POST、GET

不管 POST 或 GET，都用于向服务器请求某个 WEB 资源，这两种方式的区别主要表现在数据传递上，客户端通过这两种方式都可以带一些数据给服务器：

如请求方式为 GET 方式，则可以在请求的 URL 地址后以?的形式带上交给服务器的数据，多个数据之间以&进行分隔，例如：

GET /mail/1.html?name=abc&password=xyz HTTP/1.1

GET 方式的特点：在 URL 地址后附带的参数是有限制的，其数据容量不能超过 1K。

如请求方式为 POST 方式，则可以在请求的实体内容中向服务器发送数据，例如：

POST /servlet/ParamsServlet HTTP/1.1
Host:
Content-Type: application/x-www-form-urlencoded
Content-Length: 28

name=abc&password=xyz

Post 方式的特点：传送的数据量无限制。

Tip6：HTTP 请求的细节——消息头

用于 HTTP 请求中的常用头

Accept: text/html,image/*
Accept-Charset: ISO-8859-1
Accept-Encoding: gzip,compress
Accept-Language: en-us,zh-cn
Host: www.it315.org:80
If-Modified-Since: Tue, 11 Jul 2000 18:23:51 GMT
Referer: http://www.it315.org/index.jsp
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Cookie
Connection: close/Keep-Alive
Date: Tue, 11 Jul 2000 18:23:51 GMT

一个 HTTP 响应代表服务器向客户端回送的数据，它包括：

一个状态行、若干消息头、以及实体内容，其中的一些消息头和实体内容都是可选的，消息头和实体内容之间要用空行隔开。

HTTP/1.1 200 OK	状态行
Server: Microsoft-IIS/5.0 Date: Thu, 13 Jul 2000 05:46:53 GMT Content-Length: 2291 Content-Type: text/html Cache-control: private	多个消息头
	一个空行
<HTML> <BODY>	实体内容

状态行

格式： HTTP 版本号 状态码 原因叙述<CRLF>

举例：HTTP/1.1 200 OK

状态码用于表示服务器对请求的处理结果，它是一个三位的十进制数。响应状态码分为 5 类，如下所示：

状态码	含义
100～199	表示成功接收请求，要求客户端继续提交下一次请求才能完成整个处理过程
200～299	表示成功接收请求并已完成整个处理过程，常用 200
300～399	为完成请求，客户需进一步细化请求。例如，请求的资源已经移动一个新地址，常用 302、307 和 304

400～499	客户端的请求有错误，常用 404
500～599	服务器端出现错误，常用 500

12.7 Tip9：HTTP 响应细节——常用响应头

HTTP 请求中的常用响应头
Location: http://www.it315.org/index.jsp
Server:apache tomcat
Content-Encoding: gzip
Content-Length: 80
Content-Language: zh-cn
Content-Type: text/html; charset=GB2312
Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT
Refresh: 1;url=http://www.it315.org
Content-Disposition: attachment; filename=aaa.zip
Transfer-Encoding: chunked
Set-Cookie:SS=Q0=5Lb_nQ; path=/search
Expires: -1
Cache-Control: no-cache
Pragma: no-cache
Connection: close/Keep-Alive
Date: Tue, 11 Jul 2000 18:23:51 GMT

12.8 Tip10：HTTP 请求的细节—通用信息头

通用信息头指既能用于请求，又能用于响应的一些消息头。
Cache-Control: no-cache
Pragma: no-cache
Connection: close/Keep-Alive
Date: Tue, 11 Jul 2000 18:23:51 GMT

12.9 Tip11：作业

请写一篇关于 HTTP 协议的笔记，要求：
描述清楚 HTTP 请求头、响应头的格式
请求头和响应头中各个头字段的含义
如果浏览器传递给 WEB 服务器的参数内容超过 1K，应该使用那种方式发送请求消息？
请描述 200、302、304、404 和 500 等响应状态码所表示的意义。
请列举三种禁止浏览器缓存的头字段，并写出相应的设置值。

请求	
1.From:	请求发送者的 email 地址，由一些特殊的 Web 客户程序使用，浏览器不会用到它。
2.Accept:	浏览器可接受的 MIME 类型。
3.Accept-Encoding:	浏览器能够进行解码的数据编码方式，比如 gzip。
4．Accept-Language:	浏览器所希望的语言种类，当服务器能够提供一种以上的语言版本时要用到。
5．User-Agent：	

6．Referer:	包含一个 URL，用户从该 URL 代表的页面出发访问当前请求的页面。
7．Authorization:	
8．ChargeTo:	
9.Pragma:	
Accept-Charset：	浏览器可接受的字符集。
If-Modified-Since：	只有当所请求的内容在指定的日期之后又经过修改才返回它，否则返回 304“Not Modified”应答。
Cookie：	设置 cookie,这是最重要的请求头信息之一
Content-Length：	表示请求消息正文的长度。
Host：	初始 URL 中的主机和端口。
User-Agent	浏览器类型，如果 Servlet 返回的内容与浏览器类型有关则该值非常有用。
Connection	表示是否需要持久连接。如果 Servlet 看到这里的值为“Keep-Alive”，或者看到请求使用的是 HTTP 1.1（HTTP 1.1 默认进行持久连接），它就可以利用持久连接的优点，当页面包含多个元素时（例如 Applet，图片），显著地减少下载所需要的时间。要实现这一点，Servlet 需要在应答中发送一个 Content-Length 头，最简单的实现方法是：先把内容写入 ByteArrayOutputStream，然后在正式写出内容之前计算它的大小。
UA-Pixels，UA-Color，UA-OS，UA-CPU：	由某些版本的 IE 浏览器所发送的非标准的请求头，表示屏幕大小、颜色深度、操作系统和 CPU 类型。
Range	Range 头域可以请求实体的一个或者多个子范围。例如， 表示头 500 个字节：bytes=0-499 表示第二个 500 字节：bytes=500-999 表示最后 500 个字节：bytes=-500 表示 500 字节以后的范围：bytes=500- 第一个和最后一个字节：bytes=0-0,-1 同时指定几个范围：bytes=500-600,601-999 但是服务器可以忽略此请求头，如果无条件 GET 包含 Range 请求头，响应会以状态码 206（PartialContent）返回而不是以 200（OK）。

响应头字段仅用于响应消息	
Accept-Ranges	
Age	
ETag	缓存相关的头，能做到实时，其他的头只做到秒一级。
Last-Modified	文档的最后改动时间。客户可以通过 If-Modified-Since 请求头提供一个日期，该请求将被视为一个条件 GET，只有改动时间迟于指定时间的文档才会返回，否则返回一个 304（Not Modified）状态。
Location	这个头配合 302 使用，用于告诉浏览器去找哪个资源。Location 通常不是直接设置的，而是通过 HttpServletResponse 的 sendRedirect 方法，该方法同时设置状态代码为 302。
Proxy-Authenticate	
Retry-After	
Vary	
WWW-Authenticate	
Content-Type	表示后面的文档属于什么 MIME 类型。Servlet 默认为 text/plain，但通常需要显式地指定为 text/html。
Content-Encoding	文档的编码（Encode）方法。只有在解码之后才可以得到 Content-Type 头指定

	的内容类型。利用 gzip 压缩文档能够显著地减少 HTML 文档的下载时间。Java 的 GZIPOutputStream 可以很方便地进行 gzip 压缩，但只有 Unix 上的 Netscape 和 Windows 上的 IE 4、IE 5 才支持它。
Cache-Control	no-cache
Pragma	no-cache
Expires	应该在什么时候认为文档已经过期，从而不再缓存它？-1 或 0 不缓存
Refresh	表示浏览器应该在多少时间之后刷新文档，以秒计。除了刷新当前文档之外，你还可以通过 setHeader("Refresh", "5; URL=http://host/path")让浏览器读取指定的页面。
Content-Disposition	以下载方式打开
Transfer-Encoding	数据的传送方式
Accept-Range	源服务器如果接受字节范围请求(byte-range request)那么可以发送 Accept-Ranges: bytes 但是没有必要这样做。客户端在没有接收此头域时也可以产生字节范围请求（byte-range request）。服务器如果不能接受任何类型的范围请求（range request），将会发送 Accept-Ranges：none 去告诉客户不要尝试范围请求（range request）。
Content-Range	指定了返回的 Web 资源的字节范围。格式是：Content-Range:1000-3000/5000

数据压缩数据

```
String data="aaaa";
ByteArrayOutputStream bout=new ByteArrayOutputStream();
GZIPOutputStream gout=new GZIPOutputStream(bout);
gout.write(data.getBytes());//包装流一般有缓冲
gout.close();//为确保有数据
byte gzip[]=bout.toByteArray();//得到压缩后的数据
response.setHeader("Content-Encoding", "gzip");
response.setHeader("Content-Length", gzip.length+"");

response.getOutputStream().write(gzip);
```

显示图片

```
response.setHeader("content-type", "image/bmp");
InputStream in=this.getServletContext().getResourceAsStream("/1.bmp");
int len=0;
byte buffer[]=new byte[1024];
OutputStream out=response.getOutputStream();
while((len=in.read(buffer))>0){
    out.write(buffer,0,len);
}
```

以下载方式打开

```
response.setHeader("content-disposition", "attachment;filename=3.jpg");
InputStream in=this.getServletContext().getResourceAsStream("/1.bmp");
int len=0;
byte buffer[]=new byte[1024];
OutputStream out=response.getOutputStream();
```

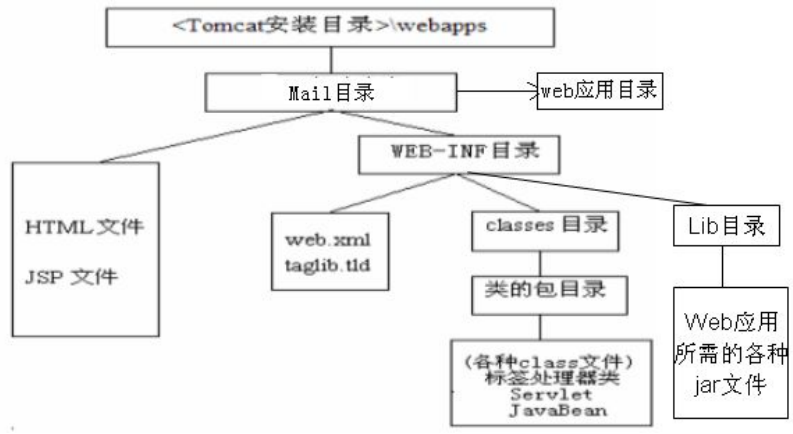
```
while((len=in.read(buffer))>0){
    out.write(buffer,0,len);
}
```

13 Servlet 开发

13.1 Tip：Servlet 简介

Servlet 是 sun 公司提供的一门用于开发动态 web 资源的技术。
Sun 公司在其 API 中提供了一个 servlet 接口，用户若想用发一个动态 web 资源(即开发一个 Java 程序向浏览器输出数据)，需要完成以下 2 个步骤：
编写一个 Java 类，实现 servlet 接口。
把开发好的 Java 类部署到 web 服务器中。
编写一个 Java 程序，向浏览器输出“hello servlet”。
编写这个程序需要解决的 2 个问题：
在 Java 程序中，如何才能向 IE 浏览器输出数据？
输出 hello servlet 的 java 代码应该写在程序的哪个方法内？
阅读 Servlet API，解决以上两个问题。

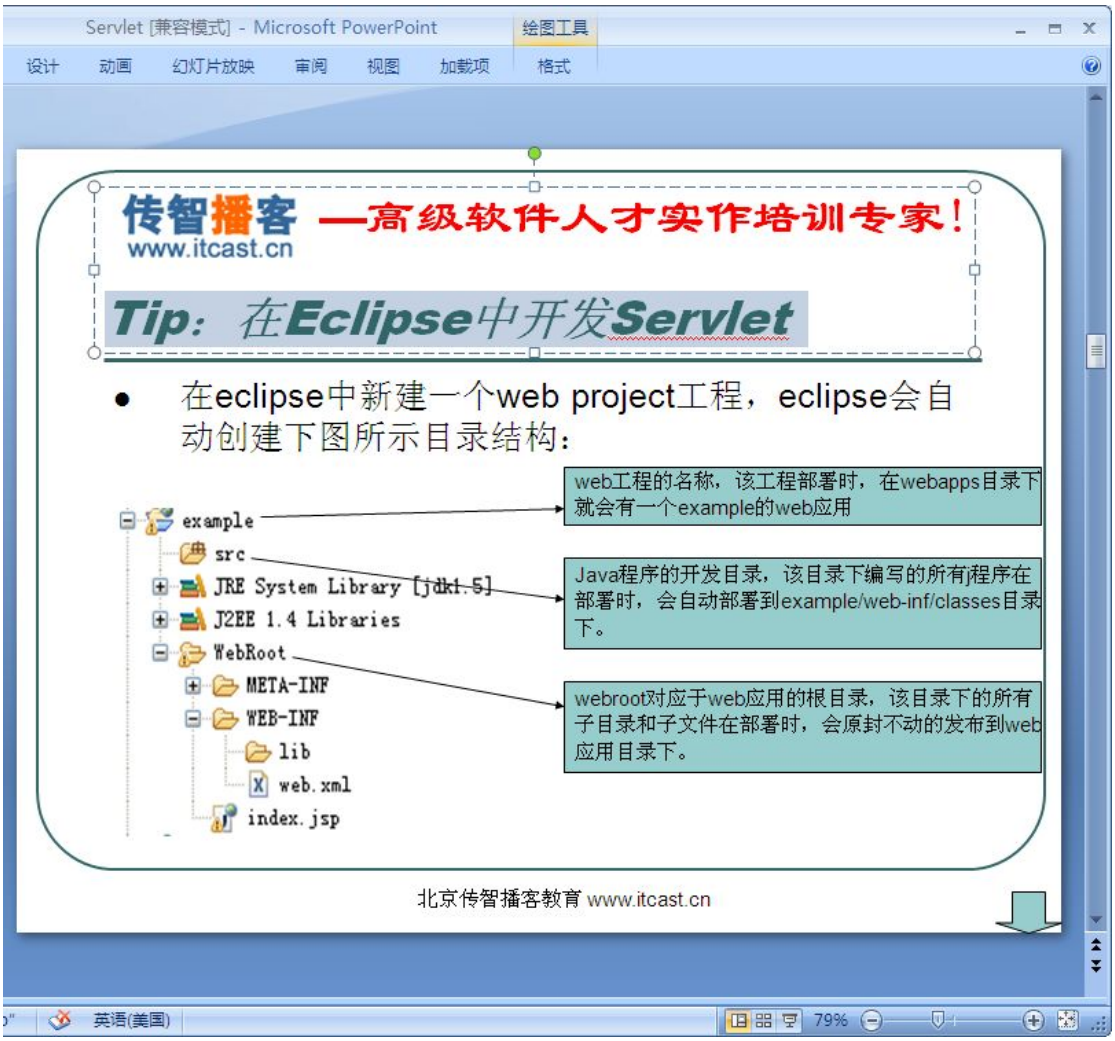
13.2 Servlet 在 web 应用中的位置



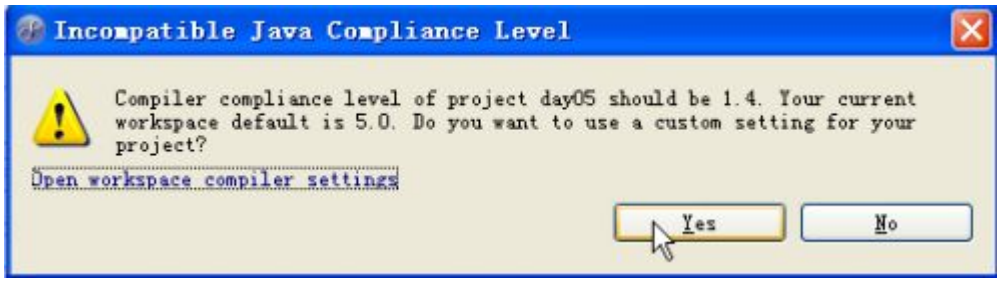
13.3 Tip：Servlet 的运行过程(课后看)

Servlet 程序是由 WEB 服务器调用，web 服务器收到客户端的 Servlet 访问请求后：
①Web 服务器首先检查是否已经装载并创建了该 Servlet 的实例对象。如果是，则直接执行第④步，否则，执行第②步。
②装载并创建该 Servlet 的一个实例对象。
③调用 Servlet 实例对象的 init()方法。
④创建一个用于封装 HTTP 请求消息的 HttpServletRequest 对象和一个代表 HTTP 响应消息的 HttpServletResponse 对象，然后调用 Servlet 的 service()方法并将请求和响应对象作为参数传递进去。
⑤WEB 应用程序被停止或重新启动之前，Servlet 引擎将卸载 Servlet，并在卸载之前调用 Servlet 的 destroy()方法。

13.4 Tip：在 Eclipse 中开发 Servlet



新建一个 j2ee 1.4 的工程，完成后说 Incompatible java Compliance Level 点 No，用新的（高版本）编译器，



添加一个 tomcat 服务器 window/Preference/MyEclipse/Servers/Timcat/Tomcat 7.x 选择目录和 Enable

更改模板刚换上 Myeclipse9.0,结果要修改 servlet 模板的时候不像 Myeclpse6.5 一样能搜索的到 servlet.java 了.
网上搜了下也没有搜到,还好求助了下老师,方法如下.
在 x:\Program Files\MyEclipse\Common\plugins 下找到 com.genuitec.eclipse.wizards_9.0.0.me201012172208.jar,然后用 winrar 打开,找到 templates 打开后就能找到 Servlet.java 了.
要是您的是 8.5 的话,搜:com.genuitec.eclipse.wizard*.jar 这个关键词就行了.
拷贝一个工程后，更改发布路径，工程/属性/MyEclipse/Web 改发布路径

13.5 Tip：Servlet 接口实现类

Servlet 接口 SUN 公司定义了两个默认实现类，分别为：GenericServlet、HttpServlet。

HttpServlet 指能够处理 HTTP 请求的 servlet，它在原有 Servlet 接口上添加了一些与 HTTP 协议处理方法，它比 Servlet 接口的功能更为强大。因此开发人员在编写 Servlet 时，通常应继承这个类，而避免直接去实现 Servlet 接口。

HttpServlet 在实现 Servlet 接口时，覆写了 service 方法，该方法体内的代码会自动判断用户的请求方式，如为 GET 请求，则调用 HttpServlet 的 doGet 方法，如为 Post 请求，则调用 doPost 方法。因此，开发人员在编写 Servlet 时，通常只需要覆写 doGet 或 doPost 方法，而不要去覆写 service 方法。

阅读 HttpServlet API 文档

13.6 Tip：Servlet 的一些细节(1)

由于客户端是通过 URL 地址访问 web 服务器中的资源，所以 Servlet 程序若想被外界访问，必须把 servlet 程序映射到一个 URL 地址上，这个工作在 web.xml 文件中使用<servlet>元素和<servlet-mapping>元素完成。
<servlet>元素用于注册 Servlet，它包含有两个主要的子元素：<servlet-name>和<servlet-class>，分别用于设置 Servlet 的注册名称和 Servlet 的完整类名。
一个<servlet-mapping>元素用于映射一个已注册的 Servlet 的一个对外访问路径，它包含有两个子元素：<servlet-name>和<url-pattern>，分别用于指定 Servlet 的注册名称和 Servlet 的对外访问路径。例如：

```
<web-app>
    <servlet>
        <servlet-name>AnyName</servlet-name>
        <servlet-class>HelloServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>AnyName</servlet-name>
        <url-pattern>/demo/hello.html</url-pattern>
    </servlet-mapping>
</web-app>
```

同一个 Servlet 可以被映射到多个 URL 上，即多个<servlet-mapping>元素的<servlet-name>子元素的设置值可以是同一个 Servlet 的注册名。
在 Servlet 映射到的 URL 中也可以使用*通配符，但是只能有两种固定的格式：一种格式是“*.扩展名”，另一种格式是以正斜杠（/）开头并以“/*”结尾。

```
<servlet-mapping>
    <servlet-name>
        AnyName
    </servlet-name>
    <url-pattern>
        *.do
    </url-pattern>
</servlet-mapping>
```

```
<servlet-mapping>
    <servlet-name>
        AnyName
    </servlet-name>
    <url-pattern>
        /action/*
    </url-pattern>
</servlet-mapping>
```

对于如下的一些映射关系：
Servlet1 映射到 /abc/*
Servlet2 映射到 /*
Servlet3 映射到 /abc
Servlet4 映射到 *.do
问题：

URL	都匹配	哪个 servlet 响应

/abc/a.html	“**/abc/***”和“/*”	Servlet1
/abc	“/abc/*”和“/abc”	Servlet3
/abc/a.do	“/abc/*”和“*.do”	Servlet1
/a.do	“**/***”和“*.do”	Servlet2
/xxx/yyy/a.do	“**/***”和“*.do”	Servlet2

Servlet 是一个供其他 Java 程序（Servlet 引擎）调用的 Java 类，它不能独立运行，它的运行完全由 Servlet 引擎来控制和调度。

针对客户端的多次 Servlet 请求，通常情况下，服务器只会创建一个 Servlet 实例对象，也就是说 Servlet 实例对象一旦创建，它就会驻留在内存中，为后续的其它请求服务，直至 web 容器退出，servlet 实例对象才会销毁。

在 Servlet 的整个生命周期内，Servlet 的 init 方法只被调用一次。而对一个 Servlet 的每次访问请求都导致 Servlet 引擎调用一次 servlet 的 service 方法。对于每次访问请求，Servlet 引擎都会创建一个新的 HttpServletRequest 请求对象和一个新的 HttpServletResponse 响应对象，然后将这两个对象作为参数传递给它调用的 Servlet 的 service()方法，service 方法再根据请求方式分别调用 doXXX 方法。
如果在<servlet>元素中配置了一个<load-on-startup>元素，那么 WEB 应用程序在启动时，就会装载并创建 Servlet 的实例对象、以及调用 Servlet 实例对象的 init()方法。

举例：

```
<servlet>
    <servlet-name>invoker</servlet-name>
    <servlet-class>
        org.apache.catalina.servlets.InvokerServlet
    </servlet-class>
    <load-on-startup>2</load-on-startup>
</servlet>
```

是一个正整数，数字越小，先启动。
用途：为 web 应用写一个 InitServlet，这个 servlet 配置为启动时装载，为整个 web 应用创建必要的数据库表和数据。
如果某个 Servlet 的映射路径仅仅为一个正斜杠（/），那么这个 Servlet 就成为当前 Web 应用程序的缺省 Servlet。
凡是在 web.xml 文件中找不到匹配的<servlet-mapping>元素的 URL，它们的访问请求都将交给缺省 Servlet 处理，也就是说，缺省 Servlet 用于处理所有其他 Servlet 都不处理的访问请求。
在<tomcat 的安装目录>\conf\web.xml 文件中，注册了一个名称为 org.apache.catalina.servlets.DefaultServlet 的 Servlet，并将这个 Servlet 设置为了缺省 Servlet。
当访问 Tomcat 服务器中的某个静态 HTML 文件和图片时，实际上是在访问这个缺省 Servlet。

13.7 Tip：Servlet 的一些细节(7)—线程安全

当多个客户端并发访问同一个 Servlet 时，web 服务器会为每一个客户端的访问请求创建一个线程，并在这个线程上调用 Servlet 的 service 方法，因此 service 方法内如果访问了同一个资源的话，就有可能引发线程安全问题。
如果某个 Servlet 实现了 SingleThreadModel 接口，那么 Servlet 引擎将以单线程模式来调用其 service 方法。
SingleThreadModel 接口中没有定义任何方法，只要在 Servlet 类的定义中增加实现 SingleThreadModel 接口的声明即可。
对于实现了 SingleThreadModel 接口的 Servlet，Servlet 引擎仍然支持对该 Servlet 的多线程并发访问，其采用的方式是产生多个 Servlet 实例对象，并发的每个线程分别调用一个独立的 Servlet 实例对象。
实现 SingleThreadModel 接口并不能真正解决 Servlet 的线程安全问题，因为 Servlet 引擎会创建多个 Servlet 实例对象，而真正意义上解决多线程安全问题是指一个 Servlet 实例对象被多个线程同时调用的问题。事实上，在 Servlet

API 2.4 中，已经将 SingleThreadModel 标记为 Deprecated（过时的）。
//子类在覆盖福雷的方法是，不能抛出比父类更多的异常

13.8 Tip：ServletConfig 对象

在 Servlet 的配置文件中，可以使用一个或多个<init-param>标签为 servlet 配置一些初始化参数。
当 servlet 配置了初始化参数后，web 容器在创建 servlet 实例对象时，会自动将这些初始化参数封装到 ServletConfig 对象中，并在调用 servlet 的 init 方法时，将 ServletConfig 对象传递给 servlet。进而，程序员通过 ServletConfig 对象就可以得到当前 servlet 的初始化参数信息。
阅读 ServletConfig API，并举例说明该对象的作用：
获得字符集编码
获得数据库连接信息
获得配置文件，查看 struts 案例的 web.xml 文件

13.9 Tip：ServletContext

WEB 容器在启动时，它会为每个 WEB 应用程序都创建一个对应的 ServletContext 对象，它代表当前 web 应用。

ServletContext 对象被包含在 ServletConfig 对象中，开发人员在编写 servlet 时，可以通过 ServletConfig.getServletContext 方法获得对 ServletContext 对象的引用。

由于一个 WEB 应用中的所有 Servlet 共享同一个 ServletContext 对象，因此 Servlet 对象之间可以通过 ServletContext 对象来实现通讯。ServletContext 对象通常也被称之为 context 域对象。

查看 ServletContext API 文档，了解 ServletContext 对象的功能。

```
package cn.itcast;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
//ServletConfig 用于封装 servlet 的配置
public class ServletDemo5 extends HttpServlet {
	private ServletConfig	config;
	public void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		String value=this.getServletConfig().getInitParameter("data");
		//通常用 value 方法，因为父类已经写了
		String value2=config.getInitParameter("data");
		System.out.println(value+","+value2);
		response.getOutputStream().write("ServeltDemo2!!".getBytes());
		//得到所有的
		Enumeration e= this.getInitParameterNames();
		while(e.hasMoreElements()){
			String a=(String) e.nextElement();
			String v=this.getServletConfig().getInitParameter(a);
			System.out.println(a+"="+v);
		}

	}
	@Override
	public void init(ServletConfig config) throws ServletException {
		this.config=config;
		super.init(config);
	}
}
```

13.10 Tip：ServletContext 应用

多个 Servlet 通过 ServletContext 对象实现数据共享。

```
package cn.itcast;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class ServletDemo6 extends HttpServlet {
//servletContext 示例
public void doGet(HttpServletRequest request,
HttpServletResponse response)
```

```
package cn.itcast;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/*
 servletContext 域：1，是一个容器 2。作用范围是应用程序范围
 */
public class ServletDemo8 extends HttpServlet {
```

```
    throws ServletException, IOException {
        //得到 servletContext
    ServletContext
context=this.getServletConfig().getServletContex
t();
    ServletContext
context2=this.getServletContext();
    String data="aaaaa";
    context.setAttribute("data", data);
    }
}
```

```
    public void doGet(HttpServletRequest request,
            HttpServletResponse response)
                throws ServletException, IOException {
        String value = (String)
this.getServletContext().getAttribute("data");

     response.getOutputStream().write(value.getBytes
());
    }
}
```

获取 WEB 应用的初始化参数。

```
<context-param>
    <param-name>data</param-name>
    <param-value>zzzz</param-value>
</context-param>
```

```
this.getServletContext().getAttribute("data");
```

实现 Servlet 的转发。

```
RequestDispatcher rd=this.getServletContext().getRequestDispatcher("/1.jsp");
    rd.forward(request, response);
```

利用 ServletContext 对象读取资源文件。
得到文件路径
读取资源文件的三种方式
.properties 文件（属性文件）

```
url=jdbc:mysql://localhost:3306/test
username=root
password=root
```

db.properties 放在 src 目录下,因为该目录下的资源文件发布时会发到/WEB-INF/classes 目录下

```
InputStreamin=this.getServletContext().getResourceAsStream("/WEB-INF/classes/db.properties");
Properties props=new Properties();
props.load(in);
System.out.println(props.getProperty("url"));
```

```
//in2 的相对路径是 tomcat 的 bin 目录，所以这种方法要在该目录下建立文件夹 classes，并把文件放在这里
        File file=new File("");
        response.getOutputStream().write(file.getAbsolutePath().getBytes());
        //以上两行代码得到当前路径 C:\apache-tomcat-7.0.22\bin
        FileInputStream in2=new FileInputStream("classes/db.properties");
        Properties props=new Properties();
        props.load(in2);
```

```
        //得到绝对路径
        String path=this.getServletContext().getRealPath("/WEB-INF/classes/db.properties");
```

```
        String filename=path.substring(path.lastIndexOf("\\"));
        FileInputStream in2=new FileInputStream(path);
        Properties props=new Properties();
        props.load(in2);
```

```
package cn.itcast;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import cn.itcast.dao.UserDao;

//servlet 调用其它程序，在其它程序中如何去读取配置文件
//通过类装载器
public class ServletDemo12 extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        UserDao user=new UserDao();
        user.update();
    }
}
```

```
package cn.itcast.dao;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.Properties;

//如果读取资源文件的程序不是 servlet 的话，
//就只能通过类转载器去读了，文件不能太大
//用传递参数方法不好，耦合性高
public class UserDao {

    private static Properties dbconfig=new Properties();
    static {
        InputStream in=UserDao.class.getClassLoader().getResourceAsStream("db.properties");
        try {
            dbconfig.load(in);
        } catch (IOException e) {
            throw new ExceptionInInitializerError(e);
        }
        //上面代码类装载器只装载一次,下面代码用类装载方式得到文件位置
        URL url=UserDao.class.getClassLoader().getResource("db.properties");
        String str=url.getPath();

    //file:/C:/apache-tomcat-7.0.22/webapps/day05/WEB-INF/classes/db.properties
        try {
            InputStream in2=new FileInputStream(str);
            try {
                dbconfig.load(in2);
            } catch (IOException e) {
                throw new ExceptionInInitializerError(e);
            }
```

```
            } catch (FileNotFoundException e1) {
                throw new ExceptionInInitializerError(e1);
            }
        }
        public void update() {
            System.out.println(dbconfig.get("url"));
        }
}
```

13.11 Tip：Servlet 高级应用—Servlet 与缓存

设置缓存的两种场景：
场景一：对于不经常变化的数据，在 servlet 中可以为其设置合理的缓存时间值，以避免浏览器频繁向服务器发送请求，提升服务器的性能。
场景二：如果要实现一种高级功能，即客户端请求动态 web 资源时，动态 web 资源发现发给客户端的数据更新了，就给客户端发送最新的数据，如果发现数据没有更新，则动态 web 资源就要客户端就去访问它自己缓存的数据。此种情况可以通过覆写动态 web 资源(即 servlet)的 getLastModify 方法予以实现。
getLastModified 方法由 service 方法调用，默认情况下，getLastModified 方法返回一个负数，开发人员在编写 servlet 时，如果不覆盖 getLastModified 方法，则每次访问 servlet 时，service 方法发现 getLastModified 方法返回负数，它就会调用 doXXX 方法向客户端返回最新的数据。此种情况下，服务器在向客户端返回 doXXX 方法返回的数据时，不会在数据上加 Last-Modified 头字段。

如果编写 servlet 时，覆盖了 getLastModified 方法，并返回某一个时间值，则客户端访问 Servlet 时，service 方法首先会检查客户端是否通过 If-Modified-Since 头字段带一个时间值过来。如果没有的话，则 service 方法会调用 doXXX 方法向客户端返回最新的数据。在返回数据时，service 方法还会调用 getLastModified 方法得到一个时间值，并以这个时间值在数据上加上一个 Last-Modified 头字段。（即通知客户端缓存数据）

客户端在访问 servlet 时，如果通过 If-Modified-Since 头字段带了一个时间值过来，则 service 方法在调用 doXXX 方法之前，它会先调用 getLastModified 方法，得到一个时间值，并与客户端带过来的时间值进行比较，如果比客户端的时间值要新，则 service 方法调用 doXXX 方法向客户端返回最新的数据。如果要旧，则 service 方法而不会调用 doXXX 方法向客户端返回数据，而是返回一个 304 的状态码给客户端，通知客户端在拿它缓存中的数据。

13.12 Tip：getLastModified 方法与缓存

13.13 Tip：缓存的应用

一个网站有很多静态资源，例如 css 文件、html 页面、gif 图片等等，这些文件一旦创建，有可能永远不会更新。当客户端第一次访问这些文件时，服务器在把文件数据交给客户端的同时，就应该通知客户端缓存这些文件，以后客户端每次访问，服务器如果发现文件没更新，则应要客户端去拿它缓存中的文件，以减轻服务器的压力。

编程：使用一个 servlet 读取一个文件数据给客户端，当文件数据未更新时，通知客户端去访问它缓存中的数据，如果文件数据更新了，则向客户端返回最新数据。

Tomcat 服务器中的所有静态 web 资源，都是由一个缺省 servlet 负责读取回送给客户端的，它就是以上方式来提升服务器的性能。

13.14 response、request 对象

Web 服务器收到客户端的 http 请求，会针对每一次请求，分别创建一个用于代表请求的 request 对象、和代表响应的 response 对象。

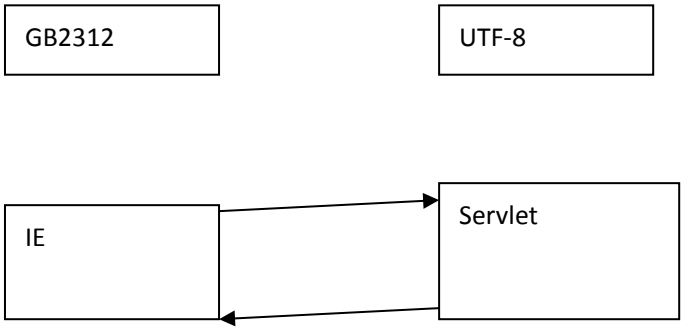
request 和 response 对象即然代表请求和响应，那我们要获取客户机提交过来的数据，只需要找 request 对象就行了。要向客户机输出数据，只需要找 response 对象就行了。

HttpServletResponse 对象服务器的响应。这个对象中封装了向客户端发送数据、发送响应头，发送响应状态码的方法。

setHeader(java.lang.String name, java.lang.String value) Sets a response header with the given name and value.	
setStatus(int sc)	Sets the status code for this response.
getWriter()	Returns a PrintWriter object that can send character text to the client.
getOutputStream()	Returns a ServletOutputStream suitable for writing binary data in the response.

Tip：response 常见应用

向客户端输出中文数据



分别以 OutputStream 和 PrintWriter 输出

```
String data="中国 ";
//程序以什么码表输出，一定要控制浏览器以什么码表打开
response.setHeader("Content-type", "text/html;charset=UTF-8");
OutputStream out=response.getOutputStream();
//out.write(data.getBytes());
out.write(data.getBytes("UTF-8"));
```

```
String data="中国 2 ";
//html: <meta>标签模拟一个 http 响应头
OutputStream out=response.getOutputStream();
//out.write(data.getBytes());
out.write("<meta http-equiv='content-type' content='text/html;charset=UTF-8'>".getBytes());
out.write(data.getBytes("UTF-8"));
```

```
String data="中国 q";
//设置 response 使用的码表，以控制 response 以什么码表向浏览器输入数据,默认为 ISO-8859-1
response.setCharacterEncoding("GB2312");
//指定浏览器以什么码表打开服务器发送的数据
response.setHeader("Content-type", "text/html;charset=GB2312");
//response.setHeader("Content-type", "text/html;charset=ISO-8859-1");
PrintWriter out=response.getWriter();
out.write(data);//
```

```
String data="中国 q";
//该句代码等价于其后面两句
response.setContentType("text/html;charset=GB2312");
//response.setCharacterEncoding("GB2312");
```

```
//response.setHeader("Content-type", "text/html;charset=ISO-8859-1");
PrintWriter out=response.getWriter();
out.write(data);//
```

用 OutputStream 输出 1，客户端看到的是什么？

13.15 文件下载

```
String path = this.getServletContext().getRealPath("/download/截图.png");
        String filename=path.substring(path.lastIndexOf("\\")+1);

        response.setHeader("content-disposition",
"attachment;filename="+URLEncoder.encode(filename,"utf-8"));
        InputStream in = null;
        OutputStream out =null;
        try{
            in = new FileInputStream(path);
            int lenth=0;
            byte buffer[]=new byte[1024];
            out = response.getOutputStream();
            while((lenth=in.read(buffer))>0){
                out.write(buffer, 0, lenth);
            }
        }finally {
            if(in!=null){
                try{
                    in.close();
                }catch (Exception e){
                    e.printStackTrace();
                }
                if(out != null){
                    try{
                        out.close();
                    }catch (Exception e) {
                        e.printStackTrace();
                    }
                }
            }
        }
```

生成随机图片

13.16 发送 http 头，控制浏览器定时刷新网页(REFRESH)

```
response.setHeader("refresh", "3");
String data = new Random().nextInt(100000)+"";
response.getWriter().write(data);
```

```
response.setHeader("refresh", "3;url='/day06/index.jsp'");
response.setContentType("text/html;charset=GB2312");
String data = new Random().nextInt(100000)+"";
response.getWriter().write("登录成功，将在 3 秒后跳转，如果没有，请点<a href=''>超链接</a>");
```

```
String message="<meta http-equiv='refresh' content='3;url=/day06/index.jsp'>登录成功，将在 3 秒后跳转，如果没有，请点<a href=''>超链接</a>";
request.setAttribute("message",message);
this.getServletContext().getRequestDispatcher("/message.jsp").forward(request, response);
```

13.17 发送 http 头，控制浏览器缓存当前文档内容

```
String  data="aaaaa";
response.setDateHeader("expires", System.currentTimeMillis()+3600*1000);
response.getWriter().write(data);
```

多学一招：使用ＨＴＭＬ语言里面的<meta>标签来控制浏览器行为

13.18 通过 response 实现请求重定向。

```
response.setStatus(302);// 302 状态码和 location 头即可实现重定向
response.setHeader("location", "/day06/index.jsp");
```

```
response.sendRedirect("/day06/index.jsp");
```

请求重定向指：一个 web 资源收到客户端请求后，通知客户端去访问另外一个 web 资源，这称之为请求重定向。
应用场景：用户注册，让用户知道注册成功，重定向到首页。
购物网站购完后，转到购物车显示页面，用转发按刷新又买一个，所以用重定向。

13.19 Tip： response 细节

getOutputStream 和 getWriter 方法分别用于得到输出二进制数据、输出文本数据的 ServletOuputStream、Printwriter 对象。

getOutputStream 和 getWriter 这两个方法互相排斥，调用了其中的任何一个方法后，就不能再调用另一方法，包括转发时。

Servlet 程序向 ServletOutputStream 或 PrintWriter 对象中写入的数据将被 Servlet 引擎从 response 里面获取，Servlet 引擎将这些数据当作响应消息的正文，然后再与响应状态行和各响应头组合后输出到客户端。

Serlvet 的 service 方法结束后，Servlet 引擎将检查 getWriter 或 getOutputStream 方法返回的输出流对象是否已经调用过 close 方法，如果没有，Servlet 引擎将调用 close 方法关闭该输出流对象。

13.20 HttpServletRequest

HttpServletRequest 对象代表客户端的请求，当客户端通过 HTTP 协议访问服务器时，HTTP 请求头中的所有信息都封装在这个对象中，开发人员通过这个对象的方法，可以获得客户这些信息。

```
getMethod()
        Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.
```

```
getParameter(java.lang.String name)
        Returns the value of a request parameter as a String, or null if the parameter does not exist.
```

```
getRequestURI()
        Returns the part of this request's URL from the protocol name up to the query string in the
```

first line of the HTTP request.
getRequestURL() Reconstructs the URL the client used to make the request.
Http://www.sina.com/news/1.html URL 子 /news/1.html URI 父

获得客户机信息

getRequestURL 方法返回客户端发出请求时的完整 URL。
getRequestURI 方法返回请求行中的资源名部分。
getQueryString 方法返回请求行中的参数部分。
getPathInfo 方法返回请求 URL 中的额外路径信息。额外路径信息是请求 URL 中的位于 Servlet 的路径之后和查询参数之前的内容，它以“/”开头。
getRemoteAddr 方法返回发出请求的客户机的 IP 地址
getRemoteHost 方法返回发出请求的客户机的完整主机名
getRemotePort 方法返回客户机所使用的网络端口号
getLocalAddr 方法返回 WEB 服务器的 IP 地址。
getLocalName 方法返回 WEB 服务器的主机名
获得客户机请求头
getHeader 方法
getHeaders 方法
getHeaderNames 方法
获得客户机请求参数(客户端提交的数据)
getParameter 方法
getParameterValues（String name）方法
getParameterNames 方法
getParameterMap 方法
Tip： request 常见应用 1
防盗链

13.21 //获取头相关数据

```
String headValue=request.getHeader("Accept-Encoding");
Enumeration e=request.getHeaderNames();
while(e.hasMoreElements()){
	String value=(String) e.nextElement();
	System.out.println(value);
}

e=request.getHeaderNames();
while(e.hasMoreElements()){
	String name=(String) e.nextElement();
	String value=request.getHeader(name);
	System.out.println("name="+name+",value="+value);
}
	}
```

```
String value=request.getParameter("username");
System.out.println(value);

Enumeration e=request.getParameterNames();
while(e.hasMoreElements()){
	String name=(String) e.nextElement();
	value=request.getParameter(name);
	System.out.println(name+"="+value);
}

String value2[]=request.getParameterValues("username");
for(String v:value2){
	System.out.println("u="+v);
}

Map<String,String[]> map=request.getParameterMap();
User user=new User();
User formbean=new User();
try {
	BeanUtils.populate(user, map);//用 map 集合填充 bean
	BeanUtils.copyProperties(user, formbean);//bean 的拷贝
} catch (Exception e1) {
	e1.printStackTrace();
}
System.out.println(user);

InputStream in=request.getInputStream();
 byte buffer[]=new byte[1024];
 int len=0;
 while((len=in.read(buffer))>0){
	System.out.println(new String(buffer,0,len));
 }
```

13.22 各种表单输入项数据的获取

text、password、radio、checkbox、file、select、textarea、 hidden、image、button 给 js 编程用

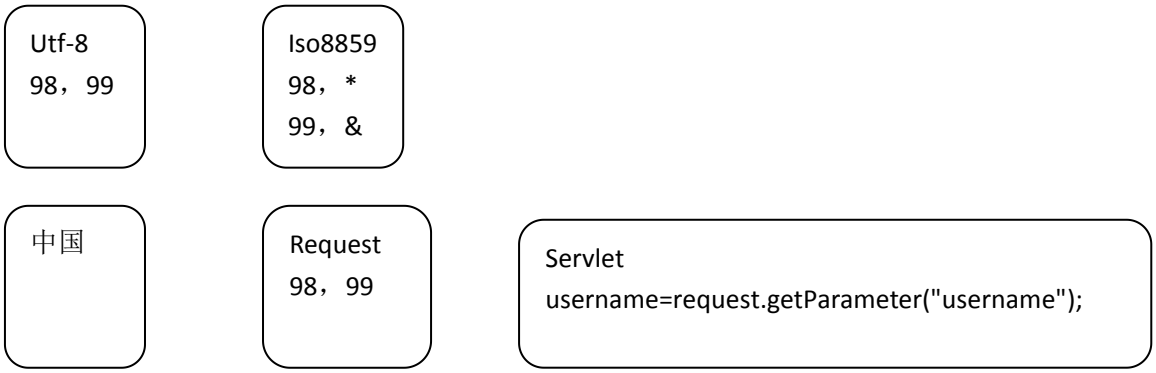
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<html>
  <head>
    <title>带数据给 requestDemo3</title>
    <meta http-equiv="content-type"
content="text/html; charset=UTF-8">
  </head>
  <body>
    <br>
```

```
request.getParameter("username");
request.getParameter("password");
request.getParameter("gender");
request.getParameter("city");
request.getParameter("city");
String[]
likes=request.getParameterValues("likes");
for(int i=0;likes!=null && i<likes.length;i++){
	System.out.println(likes[i]);
}
```

<!-- Url 后跟中文数据要编码后提交-->
<form action="/day06/servlet/requestDemo3" method="post">
用户名 1 :<input type="text" name="username"/>
 	
密码: <input type="password" name="password"/>
 	
性别:
<input type="radio" name="gender" value="male" />男
<input type="radio" name="gender" value="female" />女
 	
所在地:<select name="city">
<option value="beijing">北京</option>
<option value="shanghai">上海</option>
<option value="cs">长沙</option>
</select>
 	
爱好：
<input type="checkbox" name ="likes" value="single" />唱歌
<input type="checkbox" name ="likes" value="dance" />跳舞
<input type="checkbox" name ="likes" value="basketball" />篮球
<input type="checkbox" name ="likes" value="football" />足球
 	
备注：<textarea rows="5" cols="60" name="description" ></textarea>
 	
大头照<input type="file" name="image" />
 	
<input type="hidden" name="id" value="123456"/>
 	
<input type="submit" value="提交"/>
</form>
</body>
</html>
	request.getParameter("description");
	
	request.getParameter("id");

13.23 请求参数的中文乱码问题

浏览器已什么编码向服务器提交数据，在查看/编码下可看到，是你在做网页时指定的。



//解决 POST 乱码
request.setCharacterEncoding("UTF-8");
String username=request.getParameter("username");
System.out.println(username);
	// 解决 Get 方式提交的乱码
	String username2=request.getParameter("username2");
	username2=new String(username2.getBytes("iso-8859-1"),"UTF-8");
	System.out.println(username2);

The HTTP Connector URIEncoding This specifies the character encoding used to decode the URI bytes, after %xx decoding the URL. If not specified, ISO-8859-1 will be used.	<Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" URIEncoding="UTF-8" />
useBodyEncodingForURI This specifies if the encoding specified in contentType should be used for URI query parameters, instead of using the URIEncoding. This setting is present for compatibility with Tomcat 4.1.x, where the encoding specified in the contentType, or explicitly set using Request.setCharacterEncoding method was also used for the parameters from the URL. The default value is false.	<Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" useBodyEncodingForURI ="true" />

//无乱码
request.setCharacterEncoding("UTF-8");
String username=request.getParameter("username");
response.setCharacterEncoding("gb2312");
response.setContentType("text/htm;charset=gb2312");
response.getWriter().write(username);	

Javascript 防止表单重复提交

URL 地址的编码

request 对象实现请求转发：请求转发指一个 web 资源收到客户端请求后，通知服务器去调用另外一个 web 资源进行处理。

请求转发的应用场景：MVC 设计模式

request 对象提供了一个 getRequestDispatcher 方法，该方法返回一个 RequestDispatcher 对象，调用这个对象的 forward 方法可以实现请求转发。

request 对象同时也是一个域对象，开发人员通过 request 对象在实现转发时，把数据通过 request 对象带给其它

web 资源处理。
setAttribute 方法
getAttribute 方法
removeAttribute 方法
getAttributeNames 方法

```
//servlet 中
String message="aaaaaa";
request.setAttribute("data", message);
request.getRequestDispatcher("/mesage.jsp").forward(request, response);
```

```
Jsp 中 <%
    String mess=(String)request.getAttribute("message");
    out.write(mess);
     %>
```

13.24 Tip： 请求转发的细节

forward 方法用于将请求转发到 RequestDispatcher 对象封装的资源。
如果在调用 forward 方法之前，在 Servlet 程序中写入的部分内容已经被真正地传送到了客户端，forward 方法将抛出 IllegalStateException 异常。
如果在调用 forward 方法之前向 Servlet 引擎的缓冲区中写入了内容，只要写入到缓冲区中的内容还没有被真正输出到客户端，forward 方法就可以被正常执行，原来写入到输出缓冲区中的内容将被清空，但是，已写入到 HttpServletResponse 对象中的响应头字段信息保持有效。

13.25 Tip：请求重定向和请求转发的区别

一个 web 资源收到客户端请求后，通知服务器去调用另外一个 web 资源进行处理，称之为请求转发。
一个 web 资源收到客户端请求后，通知浏览器去访问另外一个 web 资源，称之为请求重定向。
RequestDispatcher.forward 方法只能将请求转发给同一个 WEB 应用中的组件；而 HttpServletResponse.sendRedirect 方法还可以重定向到同一个站点上的其他应用程序中的资源，甚至是使用绝对 URL 重定向到其他站点的资源。
如果传递给 HttpServletResponse.sendRedirect 方法的相对 URL 以“/”开头，它是相对于整个 WEB 站点的根目录；
如果创建 RequestDispatcher 对象时指定的相对 URL 以“/”开头，它是相对于当前 WEB 应用程序的根目录。
调用 HttpServletResponse.sendRedirect 方法重定向的访问过程结束后，浏览器地址栏中显示的 URL 会发生改变，由初始的 URL 地址变成重定向的目标 URL；调用 RequestDispatcher.forward 方法的请求转发过程结束后，浏览器地址栏保持初始的 URL 地址不变。
HttpServletResponse.sendRedirect 方法对浏览器的请求直接作出响应，响应的结果就是告诉浏览器去重新发出对另外一个 URL 的访问请求；RequestDispatcher.forward 方法在服务器端内部将请求转发给另外一个资源，浏览器只知道发出了请求并得到了响应结果，并不知道在服务器程序内部发生了转发行为。
RequestDispatcher.forward 方法的调用者与被调用者之间共享相同的 request 对象和 response 对象，它们属于同一个访问请求和响应过程；而 HttpServletResponse.sendRedirect 方法调用者与被调用者使用各自的 request 对象和 response 对象，它们属于两个独立的访问请求和响应过程。

13.26 *Tip：* RequestDispatcher

include 方法：
RequestDispatcher.include 方法用于将 RequestDispatcher 对象封装的资源内容作为当前响应内容的一部分包含进来，从而实现可编程的服务器端包含功能。
被包含的 Servlet 程序不能改变响应消息的状态码和响应头，如果它里面存在这样的语句，这些语句的执行结果将

被忽略。

```
request.getRequestDispatcher("/public/head.jsp").include(request, response);
response.getWriter().write("hahaha");
request.getRequestDispatcher("/public/foot.jsp").include(request, response);
```

```
//给服务器用 /代表当前应用
//给浏览器用 /代表网站，网站下有多个应用
request.getRequestDispatcher("/form1.html").include(request, response);
response.sendRedirect("/day06/form1.html");
this.getServletContext().getRealPath("/form1.html");
this.getServletContext().getResourceAsStream("/fomr1.html");
/**
 *
 <a href="/day06/form.html">xx</a>

 <form action="/day06/form.html">
 </form>
 */
```

```
        String referer=request.getHeader("referer");//利用 referer 防盗链
        if(referer==null && !referer.startsWith("http://localhost"))
        {
            response.sendRedirect("/day06/index.jsp");
            return;
        }
        response.getOutputStream().write("bbb".getBytes());
```

13.27 会话管理

什么是会话？

会话可简单理解为：用户开一个浏览器，点击多个超链接，访问服务器多个 web 资源，然后关闭浏览器，整个过程称之为一个会话。

会话过程中要解决的一些问题？

每个用户在使用浏览器与服务器进行会话的过程中，不可避免各自会产生一些数据，服务器要想办法为每个用户保存这些数据。

例如：多个用户点击超链接通过一个 servlet 各自购买了一个商品，服务器应该想办法把每一个用户购买的商品保存在各自的地方，以便于这些用户点结帐 servlet 时，结帐 servlet 可以得到用户各自购买的商品为用户结帐。

提问：这些数据保存在 request 或 servletContext 中行不行？

13.28 Tip：保存会话数据的两种技术：

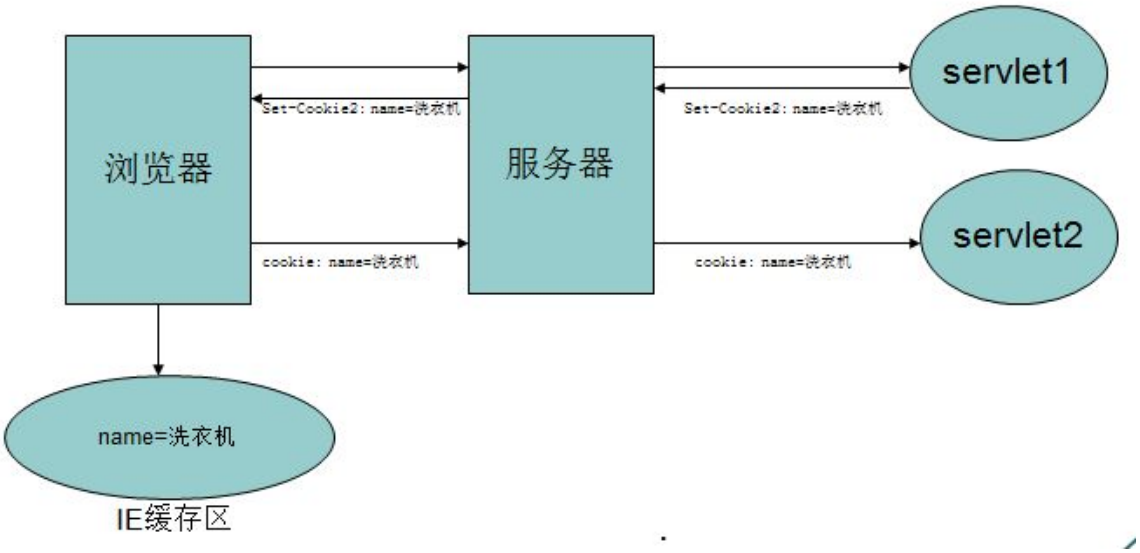
Cookie

Cookie 是客户端技术，服务器把每个用户的数据以 cookie 的形式写给用户各自的浏览器。当用户使用浏览器再去访问服务器中的 web 资源时，就会带着各自的数据去。这样，web 资源处理的就是用户各自的数据了。

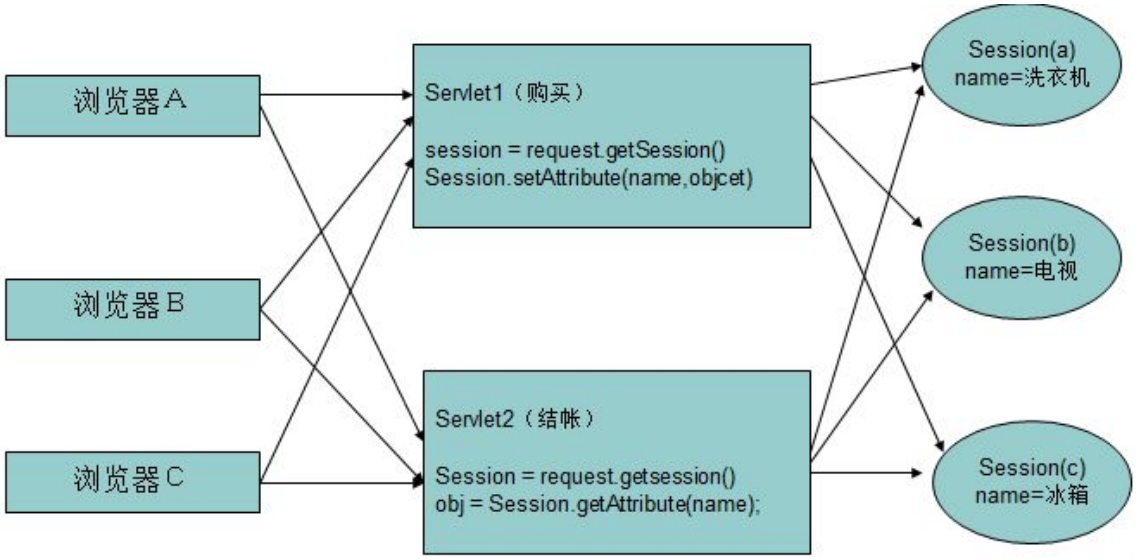
Session

Session 是服务器端技术，利用这个技术，服务器在运行时可以为每一个用户的浏览器创建一个其独享的 session 对象，由于 session 为用户浏览器独享，所以用户在访问服务器的 web 资源时，可以把各自的数据放在各自的 session 中，当用户再去访问服务器中的其它 web 资源时，其它 web 资源再从用户各自的 session 中取出数据为用户服务。

13.29 Tip：Cookie 技术



13.30 Tip：session



13.31 Tip：Cookie API

javax.servlet.http.Cookie 类用于创建一个 Cookie，response 接口也中定义了一个 addCookie 方法，它用于在其响应头中增加一个相应的 Set-Cookie 头字段。 同样，request 接口中也定义了一个 getCookies 方法，它用于获取客户端提交的 Cookie。Cookie 类的方法：

public Cookie(String name,String value)

setValue 与 getValue 方法

setMaxAge 与 getMaxAge 方法 //不调用此方法，是浏览器进程，关闭浏览器 cookie 就消失

setPath 与 getPath 方法　//day06，如果没用此方法，是谁发的 cooki，他所在的目录就带 cookie 过来
setDomain 与 getDomain 方法　.sina.com
getName 方法

13.32 Tip：Cookie 应用

显示用户上次访问时间

```
response.setCharacterEncoding("UTF-8");
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
out.print("上次访问时间：");
Cookie cookies[]=request.getCookies();
for(int i=0;cookies!=null && i<cookies.length;i++){
	if(cookies[i].getName().equals("lastAccessTime")){
		long cookieValue=Long.parseLong(cookies[i].getValue());
		Date date=new Date(cookieValue);
		out.print(date.toLocaleString());
	}
}

Cookie cookie=new Cookie("lastAccessTime",System.currentTimeMillis()+"");
cookie.setMaxAge(3600);// 以秒为单位
cookie.setPath("/day07");
response.addCookie(cookie);
```

13.33 Tip：Cookie 细节

一个 Cookie 只能标识一种信息，它至少含有一个标识该信息的名称（NAME）和设置值（VALUE）。
一个 WEB 站点可以给一个 WEB 浏览器发送多个 Cookie，一个 WEB 浏览器也可以存储多个 WEB 站点提供的 Cookie。
浏览器一般只允许存放 300 个 Cookie，每个站点最多存放 20 个 Cookie，每个 Cookie 的大小限制为 4KB。
如果创建了一个 cookie，并将他发送到浏览器，默认情况下它是一个会话级别的 cookie（即存储在浏览器的内存中），用户退出浏览器之后即被删除。若希望浏览器将该 cookie 存储在磁盘上，则需要使用 maxAge，并给出一个以秒为单位的时间。将最大时效设为 0 则是命令浏览器删除该 cookie。
注意，删除 cookie 时，path 必须一致，否则不会删除

```
//清除 cookie，用 javascript 也可实现
Cookie cookie=new Cookie("lastAccessTime",System.currentTimeMillis()+"");
cookie.setMaxAge(0);// 以秒为单位
cookie.setPath("/day07");
response.addCookie(cookie);
```

13.34 Tip：Cookie 应用

显示用户上次浏览过的商品

```
package cn.itcast.cookie;

import java.io.IOException;
```

```
import java.io.PrintWriter;
import java.util*;

import javax.servlet.*;
import javax.servlet.http.* ;

//清除 cookie
public class CookieDemo3 extends HttpServlet {

	public void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		response.setCharacterEncoding("UTF-8");
		response.setContentType("text/html;charset=UTF-8");
		PrintWriter out = response.getWriter();
		//输出所有商品
		out.write("本站有如下商品:<br/>");
		Map<String,Book>map=Db.getAll();
		for(Map.Entry<String, Book> entry:map.entrySet()){
			Book book=entry.getValue();
			out.print("<a target=\"_blank\"
href='/day07/servlet/cookieDemo4?id="+book.getId()+"'>"+book.getName()+"</a><br/>");

		}
		//显示用户看过的商品
		out.print("<br/>你曾经看过的商品<br/>");
		Cookie cookies[]=request.getCookies();
		for(int i=0;cookies!=null && i<cookies.length;i++){
			if(cookies[i].getName().equals("bookHistory")){
				String ids[]=cookies[i].getValue().split("\\,");//2,3,1
				for(String id:ids){
					Book book=(Book) Db.getAll().get(id);
					out.print("<a target=\"_blank\"
href='/day07/servlet/cookieDemo4?id="+book.getId()+"'>"+book.getName()+"</a><br/>");
				}
			}
		}
	}
}

class Db{
	private static Map<String,Book> map=new LinkedHashMap<String,Book>();
	static {
		map.put("1", new Book("1","JavaWeb 开发","老 k","一本好书"));
		map.put("2", new Book("2","jdbc 开发","老张","一本好书"));
		map.put("3", new Book("3","spring 开发","老 li","一本好书"));
		map.put("4", new Book("4","struts 开发","老张","一本好书"));
		map.put("5", new Book("5","android 开发","老 bi","一本好书"));
```

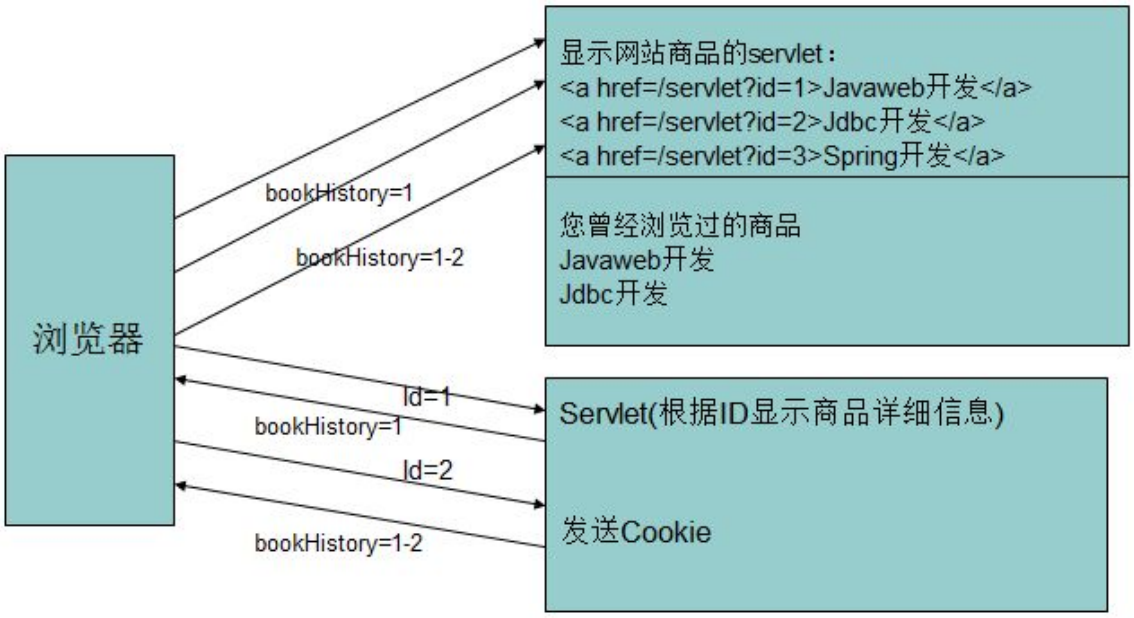
```
	}
	public static Map getAll(){
		return map;
	}
}

class Book{
	private String id;
	private String name;
	private String author;
	private String description;
	public Book() {
		super();
	}
	public Book(String id, String name, String author, String description) {
		this.id = id;
		this.name = name;
		this.author = author;
		this.description = description;
	}
	public String getId() {
		return id;
	}
	public void setId(String id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getAuthor() {
		return author;
	}
	public void setAuthor(String author) {
		this.author = author;
	}
	public String getDescription() {
		return description;
	}
	public void setDescription(String description) {
		this.description = description;
	}
}
```

```
package cn.itcast.cookie;
```

```
import java.io.*;
import java.util*;

import javax.servlet.*;
import javax.servlet.http.* ;
//显示详细信息的 servlet

public class CookieDemo4 extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        //根据用户带过来的 id，显示相应的详细信息
        response.setCharacterEncoding("UTF-8");
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String id=request.getParameter("id");
        Book book=(Book)Db.getAll().get(id);
        out.write(book.getId()+"<br/>");
        out.write(book.getName()+"<br/>");
        out.write(book.getAuthor()+"<br/>");
        out.write(book.getDescription()+"<br/>");
        //2.构建 cookie，回写给浏览器；
        String cookieValue=buildCookie(id,request);
        Cookie cookie=new Cookie("bookHistory",cookieValue);
        cookie.setMaxAge(1*30*24*3600);//1 个月
        cookie.setPath("/day07");
        response.addCookie(cookie);
    }

    private String buildCookie(String id, HttpServletRequest request) {
        //bookHistory =null     1       1
        //bookHistory=2,5,1     1       1,2,5
        //bookHistory=2,5,4     1       1,2,5
        //bookHistroy=2,5       1       1,2,5     //  假如列表最多 3 个
        String bookHistroy=null;
        Cookie cookies[]=request.getCookies();
        for(int i=0;cookies!=null && i<cookies.length;i++){
            if(cookies[i].getName().equals("bookHistory")){
                bookHistroy=cookies[i].getValue();
            }
        }
        if(bookHistroy==null)
            return id;
        //if(bookHistroy.contains(id))不能这样   21,23  也包括 1
        List<String> list=Arrays.asList(bookHistroy.split("\\,"));
        LinkedList <String>linkedlist=new LinkedList<String>(list);
```

```
            if(list.contains(id)){
                linkedlist.remove(id);
                linkedlist.addFirst(id);
            }else{
                if(list.size()>=3){
                    linkedlist.removeLast();
                    linkedlist.addFirst(id);
                }else
                    linkedlist.addFirst(id);
            }
            StringBuffer sb=new StringBuffer();
            for(String bid : linkedlist){
                sb.append(bid+",");
            }
            return sb.deleteCharAt(sb.length()-1).toString();
        }
}
```

13.35 Tip：显示上次浏览商品的实现过程



13.36 Tip：session

在 WEB 开发中，服务器可以为每个用户浏览器创建一个会话对象（session 对象），注意：一个浏览器独占一个 session 对象(默认情况下)。因此，在需要保存用户数据时，服务器程序可以把用户数据写到用户浏览器独占的 session 中，当用户使用浏览器访问其它程序时，其它程序可以从用户的 session 中取出该用户的数据，为用户服务。

Session 和 Cookie 的主要区别在于：

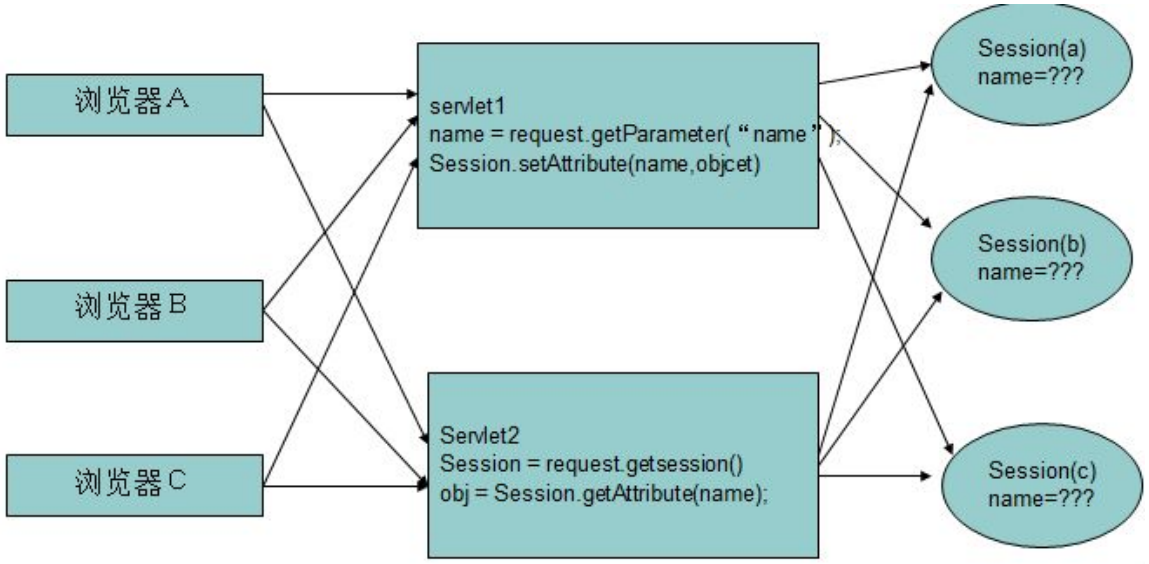
Cookie 是把用户的数据写给用户的浏览器。

Session 技术把用户的数据写到用户独占的 session 中。

Session 对象由服务器创建，开发人员可以调用 request 对象的 getSession 方法得到 session 对象。

不是访问一个页面就创建 session，而是遇到语句 request..getSession();就创建了一个 session。

注意，虽然代码相同，但不同浏览器得到的各自的数据



Session 是发呆时间，而不是累计时间

```
<session-config>
    <session-timeout>60</session-timeout>
</session-config>
```

```
HttpSession session=request.getSession();
// request.getSession(false);只获取，而不创建，通常用在查看购物车上，有些人没买商品就点查看
session.setAttribute("name", "洗衣机");
session.setMaxInactiveInterval(3600);//1 小时
session.invalidate();//摧毁
```

Session 小实验：使用 IE 访问某一个 servlet，其它 IE 可以取到这个 servlet 存的数据吗？

13.37 Tip：session 案例

使用 Session 完成简单的购物功能

```
//ListBookServlet
response.setCharacterEncoding("UTF-8");
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
out.print("本站有如下商品<br/>");
Map <String,Book>map=Db.getAll();
for(Map.Entry<String, Book> entry:map.entrySet()){
    Book book=entry.getValue();
    String str="<a target=\"_blank\"
href='"+request.getContextPath()+"/servlet/buyServlet?id="+book.getId()+"'>"+book.getName()+"购买
"+"</a><br/>";
    out.print(str);
}
System.out.println("a"+request.getContextPath()+"b");
```

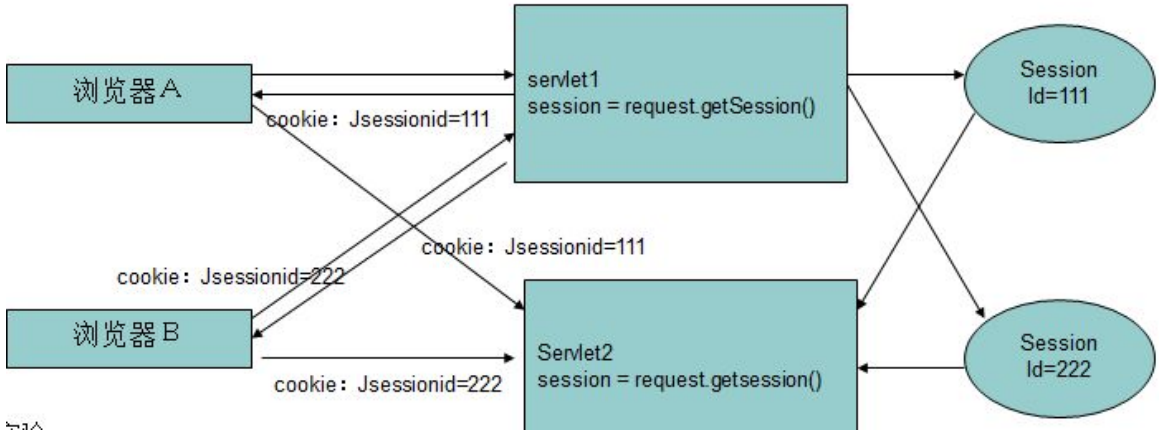
```
    //BuyServlet
response.setContentType("UTF-8");
response.setContentType("text/html;charset=UTF-8");
```

```
PrintWriter out = response.getWriter();
String id=request.getParameter("id");
Book book=(Book)Db.getAll().get(id);
HttpSession session=request.getSession();
//用 session 中得到用户购买的商品集合
List <Book> list=(List)session.getAttribute("list");
if(list==null){
    list=new LinkedList<Book>();
    session.setAttribute("list", list);
}
list.add(book);
response.sendRedirect(request.getContextPath()+"/servlet/listCartServlet");
```

```
//ListCartServlet
response.setCharacterEncoding("UTF-8");
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
HttpSession session=request.getSession(false);
if(session==null){
    out.write("您没有购买任何商品");return;
}
List<Book> list=(List) session.getAttribute("list");
out.write("你购买了如下商品");
for(Book book:list){
    out.write(book.getName()+"<br/>");
}
```

13.38 Tip：session 实现原理

疑问：服务器是如何实现一个 session 为一个用户浏览器服务的？



问题：如何实现多个 IE 浏览器共享同一 session？(应用：关掉 IE 后，再开 IE，上次购买的商品还在。)

```
HttpSession session=request.getSession();
session.setAttribute("name", "洗衣机");
```

```
String id=session.getId();
Cookie cookie=new Cookie("JSESSIONID",id);
cookie.setPath("/day07");
cookie.setMaxAge(30*60);//30 minutes
response.addCookie(cookie);
```

13.39 Tip：IE 禁用 Cookie 后的 session 处理

实验演示禁用 Cookie 后 servlet 共享数据导致的问题。

解决方案：URL 重写

response. encodeRedirectURL(java.lang.String url)

用于对 sendRedirect 方法后的 url 地址进行重写。

response. encodeURL(java.lang.String url)

用于对表单 action 和超链接的 url 地址进行重写

```
<%
   request.getSession();
   String str1=response.encodeURL("/day07/servlet/sessionDemo1");
   String str2=response.encodeURL("/day07/servlet/sessionDemo2");
%>
   <a href=<%=str1%> target="_blank">购买</a> <br>
   <a href=<%=str2 %> target="_blank">结账</a> <br>
```

附加：

Session 的失效

Web.xml 文件配置 session 失效时间

13.40 Tip：session 案例

使用 Session 完成用户登陆

利用 Session 实现一次性验证码

利用 Session 防止表单重复提交

```
<script type="text/javascript">
    var isCommited = false;
    function checkPost(){
        if(!isCommited){
            //document.getElementById("sub").disabled = true;
            isCommited = true;
            return true;
        }else{
            alert("不能重复提交");
            return false;
        }
    }
</script>
```

不足：但用户单击”刷新”，或单击”后退”再次提交表单，将导致表单重复提交

13.41 Tip：session 案例－防止表单重复提交

表单页面由 servlet 程序生成，servlet 为每次产生的表单页面分配一个唯一的随机标识号，并在 FORM 表单的一个

隐藏字段中设置这个标识号，同时在当前用户的 Session 域中保存这个标识号。
当用户提交 FORM 表单时，负责处理表单提交的 serlvet 得到表单提交的标识号，并与 session 中存储的标识号比较，如果相同则处理表单提交，处理完后清除当前用户的 Session 域中存储的标识号。
在下列情况下，服务器程序将拒绝用户提交的表单请求：
存储 Session 域中的表单标识号与表单提交的标识号不同
当前用户的 Session 中不存在表单标识号
用户提交的表单数据中没有标识号字段
编写工具类生成表单标识号：TokenProcessor

```
//首页

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        //产生随机数（表单号)
TokenProcessor tp=TokenProcessor.getInstance();
String token=tp.generateToken();
request.getSession().setAttribute("token", token);
request.getRequestDispatcher("/form.jsp").forward(request, response);
}
class TokenProcessor{//令牌
    /*
     * 1。构造 方法私有
     * 2。自己创建一个
     * 3。对外暴露一个方法，允许获取上面创建的对象      *
     */
    private TokenProcessor(){}
    private static final TokenProcessor instance=new TokenProcessor();
    public static TokenProcessor getInstance(){
        return instance;

    }
    public String generateToken(){
        String token=System.currentTimeMillis()+new Random().nextInt()+"";
        try {
            MessageDigest md=MessageDigest.getInstance("md5");
            byte []md5=md.digest(token.getBytes());
            //base64 编码
            BASE64Encoder encoder=new BASE64Encoder();//文档没有正式发布
            return encoder.encode(md5);
            //return new String(md5);
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException(e);
        }
        //return null;
    }
}
```

```
//form.jsp
```

```
<%@ page language="java" pageEncoding="UTF-8"%>


<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>

  <head>
    <title>My JSP 'form.jsp' starting page</title>

    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
  </head>
   <script type="text/javascript">
       var iscommitted=false;
       function dosubmit(){
          if(!iscommitted){
                iscommitted=true;
                document.getElementById("tbn").disabled="disabled";
                return true;
          }else{
          return false;
          }
    }
  </script>

  <body>
    <form action="/day07/servlet/doFormServlet" method="post" onsubmit="return dosubmit()">
       用户名：<input type="text" name="username"/>
       <input id="btn" type="submit" value="提交"/>
       <input name="token" type="hidden" value="${token}"/>
    </form>
  </body>
</html>
```

```
//处理提交的 servlet
      request.setCharacterEncoding("UTF-8");
/*
 * String username=request.getParameter("username"); try {
 * Thread.sleep(1000*3); } catch (InterruptedException e) { // TODO
 * Auto-generated catch block e.printStackTrace(); }
 * System.out.println("向数据库写用户名");
 */
String r_token = request.getParameter("token");
HttpSession session = request.getSession(false);
if (r_token != null
            && session != null
```

```
            && r_token.equalsIgnoreCase((String) session.getAttribute("token"))) {
        request.getSession().removeAttribute("token");
        System.out.println("向数据库写用户名");
} else {
        System.out.println("重复提交");
}
```

13.42 Tip：session 案例一次性校验码

一次性验证码的主要目的就是为了限制人们利用工具软件来暴力猜测密码。
服务器程序接收到表单数据后，首先判断用户是否填写了正确的验证码，只有该验证码与服务器端保存的验证码匹配时，服务器程序才开始正常的表单处理流程。
密码猜测工具要逐一尝试每个密码的前题条件是先输入正确的验证码，而验证码是一次性有效的，这样基本上就阻断了密码猜测工具的自动地处理过程。

Request	显示完就不用了
session	显示完等下还要用,用户登录
context	显示完等下还要用,还要给别人用,如聊天室

13.43 应用 Session+Cookie 技术完成用户自动登陆功能

14 Java Server Pages

JSP 全称是 Java Server Pages，它和 servle 技术一样，都是 SUN 公司定义的一种用于开发动态 web 资源的技术。

JSP 这门技术的最大的特点在于，写 jsp 就像在写 html，但：
它相比 html 而言，html 只能为用户提供静态数据，而 Jsp 技术允许在页面中嵌套 java 代码，为用户提供动态数据。
相比 servlet 而言，servlet 很难对数据进行排版，而 jsp 除了可以用 java 代码产生动态数据的同时，也很容易对数据进行排版。

Jsp 快速入门：在 jsp 页面中输出当前时间。

把英文版 myeclipse 8.6 中 jsp 默认编码格式改为 pageEncoding="utf-8"
windows -- > Preferences -- > MyEclipse -- > Files and Editors -- > JSP 然后右边选择 Encoding 为 ISO 10646/Unicode(UTF-8)

14.1 Tip：JSP 最佳实践

不管是 JSP 还是 Servlet，虽然都可以用于开发动态 web 资源。但由于这 2 门技术各自的特点，在长期的软件实践中，人们逐渐把 servlet 作为 web 应用中的控制器组件来使用，而把 JSP 技术作为数据显示模板来使用。

其原因为，程序的数据通常要美化后再输出：
让 jsp 既用 java 代码产生动态数据，又做美化会导致页面难以维护。
让 servlet 既产生数据，又在里面嵌套 html 代码美化数据，同样也会导致程序可读性差，难以维护。

因此最好的办法就是根据这两门技术的特点，让它们各自负责各的，servlet 只负责响应请求产生数据，并把数据通过转发技术带给 jsp，数据的显示 jsp 来做。

14.2 Tip：JSP 原理

目标：
Web 服务器是如何调用并执行一个 jsp 页面的？
Jsp 页面中的 html 排版标签是如何被发送到客户端的？
Jsp 页面中的 java 代码服务器是如何执行的？
Web 服务器在调用 jsp 时，会给 jsp 提供一些什么 java 对象？

思考：JSP 为什么可以像 servlet 一样，也可以叫做动态 web 资源的开发技术？

14.3 Tip：JSP 语法

JSP 模版元素
JSP 表达式
JSP 脚本片断
JSP 注释
JSP 指令
JSP 标签
JSP 内置对象
如何查找 JSP 页面中的错误

14.4 Tip： JSP 模版元素

JSP 页面中的 HTML 内容称之为 JSP 模版元素。
JSP 模版元素定义了网页的基本骨架，即定义了页面的结构和外观。

14.5 Tip： JSP 脚本表达式

JSP 脚本表达式（expression）用于将程序数据输出到客户端
语法：<%= 变量或表达式 %>
举例：当前时间:<%= new java.util.Date() %>
JSP 引擎在翻译脚本表达式时，会将程序数据转成字符串，然后在相应位置用 out.print(…) 将数据输给客户端。
JSP 脚本表达式中的变量或表达式后面不能有分号（;）。

14.6 Tip： JSP 脚本片断（1）

JSP 脚本片断（scriptlet）用于在 JSP 页面中编写多行 Java 代码。语法：

```
<%
        多行 java 代码
%>
```

注意：JSP 脚本片断中只能出现 java 代码，不能出现其它模板元素， JSP 引擎在翻译 JSP 页面中，会将 JSP 脚本片断中的 Java 代码将被原封不动地放到 Servlet 的_jspService 方法中。
JSP 脚本片断中的 Java 代码必须严格遵循 Java 语法，例如，每执行语句后面必须用分号（;）结束。
在一个 JSP 页面中可以有多个脚本片断，在两个或多个脚本片断之间可以嵌入文本、HTML 标记和其他 JSP 元素。
举例：

```
<%
    int x = 10;
```

```
    out.println(x);
%>
<p>这是 JSP 页面文本</p>
<%
    int y = 20;
    out.println(y);
    %>
```

多个脚本片断中的代码可以相互访问，犹如将所有的代码放在一对<%%>之中的情况。如：out.println(x);
单个脚本片断中的 Java 语句可以是不完整的，但是，多个脚本片断组合后的结果必须是完整的 Java 语句，例如：

```
<%
    for (int i=1; i<5; i++)
    {
%>

    <H1>www.it315.org</H1>

<%
    }
%>
```

14.7 Tip： JSP 声明

JSP 页面中编写的所有代码，默认会翻译到 servlet 的 service 方法中，而 Jsp 声明中的 java 代码被翻译到_jspService 方法的外面。语法：

```
<%!
    java 代码
%>
```

所以，JSP 声明可用于定义 JSP 页面转换成的 Servlet 程序的静态代码块、成员变量和方法 。

多个静态代码块、变量和函数可以定义在一个 JSP 声明中，也可以分别单独定义在多个 JSP 声明中。

JSP 隐式对象的作用范围仅限于 Servlet 的_jspService 方法，所以在 JSP 声明中不能使用这些隐式对象。

14.8 Tip： JSP 声明 案例

```
<%!
static
{
    System.out.println("loading Servlet!");
}
private int globalVar = 0;
public void jspInit()
{
    System.out.println("initializing jsp!");
}
%>
<%!
public void jspDestroy()
```

```
{
    System.out.println("destroying jsp!");
}
%>
```

14.9 Tip： JSP 注释

JSP 注释的格式：

<%-- 注释信息 --%>

JSP 引擎在将 JSP 页面翻译成 Servlet 程序时，忽略 JSP 页面中被注释的内容。

14.10 Tip： JSP 指令

JSP 指令（directive）是为 JSP 引擎而设计的，它们并不直接产生任何可见输出，而只是告诉引擎如何处理 JSP 页面中的其余部分。在 JSP 2.0 规范中共定义了三个指令：

page 指令

Include 指令

taglib 指令

14.11 Tip： JSP 指令简介

JSP 指令的基本语法格式：

<%@ 指令 属性名="值" %>

举例：<%@ page contentType="text/html;charset=gb2312"%>

如果一个指令有多个属性，这多个属性可以写在一个指令中，也可以分开写。

例如：

<%@ page contentType="text/html;charset=gb2312"%>

<%@ page import="java.util.Date"%>

也可以写作：

<%@ page contentType="text/html;charset=gb2312" import="java.util.Date"%>

14.12 Tip： Page 指令

page 指令用于定义 JSP 页面的各种属性，无论 page 指令出现在 JSP 页面中的什么地方，它作用的都是整个 JSP 页面，为了保持程序的可读性和遵循良好的编程习惯，page 指令最好是放在整个 JSP 页面的起始位置。

JSP 2.0 规范中定义的 page 指令的完整语法：

```
<%@ page
    [ language="java" ]
    [ extends="package.class" ]
    [ import="{package.class | package.*}, ..." ]
    [ session="true | false" ]
    [ buffer="none | 8kb | sizekb" ]
    [ autoFlush="true | false" ]
    [ isThreadSafe="true | false" ]
    [ info="text" ]
    [ errorPage="relative_url" ]
    [ isErrorPage="true | false" ]
    [ contentType="mimeType [ ;charset=characterSet ]" | "text/html ; charset=ISO-8859-1" ]
    [ pageEncoding="characterSet | ISO-8859-1" ]
```

[isELIgnored="true | false"]
%>
JSP 引擎自动导入下面的包：
java.lang.*
javax.servlet.*
javax.servlet.jsp.*
javax.servlet.http.*
errorPage 属性的设置值必须使用相对路径，如果以“/”开头，表示相对于当前 WEB 应用程序的根目录（注意不是站点根目录），否则，表示相对于当前页面。
可以在 web.xml 文件中使用<error-page>元素为整个 WEB 应用程序设置错误处理页面，其中的<exception-type>子元素指定异常类的完全限定名，<location>元素指定以“/”开头的错误处理页面的路径。
如果设置了某个 JSP 页面的 errorPage 属性，那么在 web.xml 文件中设置的错误处理将不对该页面起作用。

JSP 引擎会根据 page 指令的 contentType 属性生成相应的调用 ServletResponse.setContentType 方法的语句。
page 指令的 contentType 属性还具有说明 JSP 源文件的字符编码的作用。

14.13 Tip：使用 page 指令解决 JSP 中文乱码

JSP 程序存在有与 Servlet 程序完全相同的中文乱码问题
输出响应正文时出现的中文乱码问题
读取浏览器传递的参数信息时出现的中文乱码问题
JSP 引擎将 JSP 页面翻译成 Servlet 源文件时也可能导致中文乱码问题
JSP 引擎将 JSP 源文件翻译成的 Servlet 源文件默认采用 UTF-8 编码，而 JSP 开发人员可以采用各种字符集编码来编写 JSP 源文件，因此，JSP 引擎将 JSP 源文件翻译成 Servlet 源文件时，需要进行字符编码转换。
如果 JSP 文件中没有说明它采用的字符集编码，JSP 引擎将把它当作默认的 ISO8859-1 字符集编码处理。
如何解决 JSP 引擎翻译 JSP 页面时的中文乱码问题
通过 page 指令的 contentType 属性说明 JSP 源文件的字符集编码
page 指令的 pageEncoding 属性说明 JSP 源文件的字符集编码

14.14 Tip： include 指令

include 指令用于引入其它 JSP 页面，如果使用 include 指令引入了其它 JSP 页面，那么 JSP 引擎将把这两个 JSP 翻译成一个 servlet。所以 include 指令引入通常也称之为静态引入。在编译时包含
语法：
<%@ include file="relativeURL"%>
其中的 file 属性用于指定被引入文件的相对路径。 file 属性的设置值必须使用相对路径，如果以“/”开头，表示相对于当前 WEB 应用程序的根目录（注意不是站点根目录），否则，表示相对于当前文件。
细节：
被引入的文件必须遵循 JSP 语法。
被引入的文件可以使用任意的扩展名，即使其扩展名是 html，JSP 引擎也会按照处理 jsp 页面的方式处理它里面的内容，为了见明知意，JSP 规范建议使用.jspf（JSP fragments）作为静态引入文件的扩展名。
由于使用 include 指令将会涉及到 2 个 JSP 页面，并会把 2 个 JSP 翻译成一个 servlet，所以这 2 个 JSP 页面的指令不能冲突（除了 pageEncoding 和导包除外）。
request.getRequestDispatcher("/a.jsp").forward(request,response);//动态包含，翻译成多个 servlet，运行时包含

14.15 Tip： taglib 指令

Taglib 指令用于在 JSP 页面中导入标签库，讲自定义标签技术时讲。

14.16 Tip：JSP 运行原理和九大隐式对象

每个 JSP 页面在第一次被访问时，WEB 容器都会把请求交给 JSP 引擎（即一个 Java 程序）去处理。JSP 引擎先将 JSP 翻译成一个_jspServlet(实质上也是一个 servlet) ，然后按照 servlet 的调用方式进行调用。
由于 JSP 第一次访问时会翻译成 servlet，所以第一次访问通常会比较慢，但第二次访问，JSP 引擎如果发现 JSP 没有变化，就不再翻译，而是直接调用，所以程序的执行效率不会受到影响。
JSP 引擎在调用 JSP 对应的_jspServlet 时，会传递或创建 9 个与 web 开发相关的对象供_jspServlet 使用。JSP 技术的设计者为便于开发人员在编写 JSP 页面时获得这些 web 对象的引用，特意定义了 9 个相应的变量，开发人员在 JSP 页面中通过这些变量就可以快速获得这 9 大对象的引用。
这 9 个对象分别是哪些，以及作用也是笔试经常考察的知识点。
request
response
config
application
exception
Session
page
out
pageContext

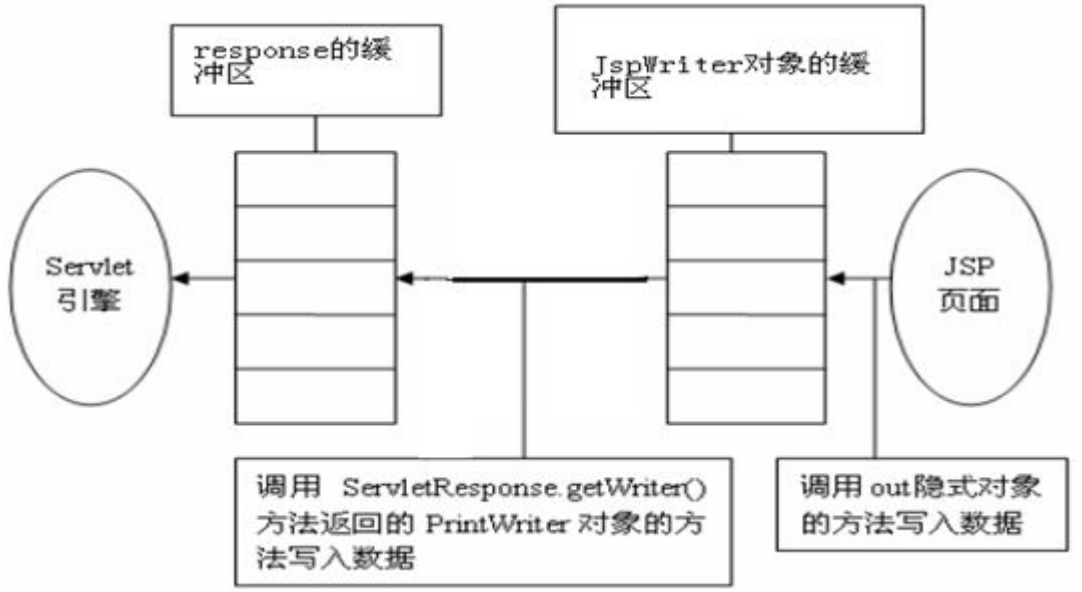
14.17 Tip：JSP 九大隐式对象对应关系

request HttpServletRequest
response HttpServletResponse
session HttpSession
application servletContext
config servletConfig
out JspWriter -----> PrintWriter
exception
page this
pageContext

14.18 Tip： out 隐式对象

out 隐式对象用于向客户端发送文本数据。
out 对象是通过调用 pageContext 对象的 getOut 方法返回的，其作用和用法与 ServletResponse.getWriter 方法返回的 PrintWriter 对象非常相似。
JSP 页面中的 out 隐式对象的类型为 JspWriter，JspWriter 相当于一种带缓存功能的 PrintWriter，设置 JSP 页面的 page 指令的 buffer 属性可以调整它的缓存大小，甚至关闭它的缓存。
只有向 out 对象中写入了内容，且满足如下任何一个条件时，out 对象才去调用 ServletResponse.getWriter 方法，并通过该方法返回的 PrintWriter 对象将 out 对象的缓冲区中的内容真正写入到 Servlet 引擎提供的缓冲区中：
设置 page 指令的 buffer 属性关闭了 out 对象的缓存功能
out 对象的缓冲区已满
整个 JSP 页面结束

14.19 Tip： out 隐式对象的工作原理图



14.20 Tip： out 隐式对象的注意事项

同时使用 out 和 response.getwriter()输出数据。
用 JSP 实现文件下载。

14.21 Tip： pageContext 对象

pageContext 对象是 JSP 技术中最重要的一个对象，它代表 JSP 页面的运行环境，这个对象不仅封装了对其它 8 大隐式对象的引用，它自身还是一个域对象，可以用来保存数据。并且，这个对象还封装了 web 开发中经常涉及到的一些常用操作，例如引入和跳转其它资源、检索其它域对象中的属性等。

14.22 Tip：通过 pageContext 获得其他对象

getException 方法返回 exception 隐式对象
getPage 方法返回 page 隐式对象
getRequest 方法返回 request 隐式对象
getResponse 方法返回 response 隐式对象
getServletConfig 方法返回 config 隐式对象
getServletContext 方法返回 application 隐式对象
getSession 方法返回 session 隐式对象
getOut 方法返回 out 隐式对象
pageContext 封装其它 8 大内置对象的意义，思考：如果在编程过程中，把 pageContext 对象传递给一个普通 java 对象，那么这个 java 对象将具有什么功能？ //用在 自定义标签上

14.23 Tip：pageContext 作为域对象

pageContext 对象的方法
public void setAttribute(java.lang.String name,java.lang.Object value)
public java.lang.Object getAttribute(java.lang.String name)
public void removeAttribute(java.lang.String name)
pageContext 对象中还封装了访问其它域的方法
public java.lang.Object getAttribute(java.lang.String name,int scope)

public void setAttribute(java.lang.String name, java.lang.Object value,int scope)
public void removeAttribute(java.lang.String name,int scope)
代表各个域的常量
PageContext.APPLICATION_SCOPE
PageContext.SESSION_SCOPE
PageContext.REQUEST_SCOPE
PageContext.PAGE_SCOPE
findAttribute 方法　　（*重点，查找各个域中的属性）

14.24 Tip：重点

到此为止，web 开发接触到了 4 个域对象：
pageContext（称之为 page 域）
request（称之为 request 域）
session（称之为 session 域）
servletContext（称之为 application 域）
这 4 个域对象是学习 web 的重点，也是笔试经常考察的知识点。
明确如下问题：
这 4 个对象的生命周期？
什么是域？为什么把这 4 个对象叫做域对象呢？
哪种情况下用哪种域对象。

14.25 Tip：引入和跳转到其他资源

PageContext 类中定义了一个 forward 方法和两个 include 方法来分别简化和替代 RequestDispatcher.forward 方法和 include 方法

传递给这些方法的资源路径都只能是相对路径，如果路径以“/”开头，表示相对于当前 WEB 应用程序的根目录，否则，表示相对于当前 JSP 所映射到的访问路径。

14.26 Tip： JSP 标签

虽然我们希望 JSP 页面仅用作数据显示模块，不要嵌套任何 java 代码引入任何业务逻辑，但在实际开发中不引入一点业务逻辑是不可能的，但引入业务逻辑会导致页面出现难看 java 代码，怎么办？
Sun 公司允许用户开发自定义标签封装页面的 java 代码，以便 jsp 页面不出现一行 java 代码。当然 sun 公司在 jsp 页面中也内置了一些标签(这些标签叫做 jsp 标签)，开发人员使用这些标签可以完成页面的一些常用业务逻辑。
JSP 标签也称之为 Jsp Action(JSP 动作)元素，它用于在 JSP 页面中提供业务逻辑功能。

14.27 Tip： JSP 常用标签

<jsp:include>标签
<jsp:forward>标签
<jsp:param>标签

14.28 Tip： <jsp:include>标签

<jsp:include>标签用于把另外一个资源的输出内容插入进当前 JSP 页面的输出内容之中，这种在 JSP 页面执行时的引入方式称之为动态引入。
语法：

```
<jsp:include page="relativeURL | <%=expression%>" flush="true|false" />
```

page 属性用于指定被引入资源的相对路径，它也可以通过执行一个表达式来获得。
flush 属性指定在插入其他资源的输出内容时，是否先将当前 JSP 页面的已输出的内容刷新到客户端。

14.29 Tip： <jsp:include>与 include 指令的比较

<jsp:include>标签是动态引入， <jsp:include>标签涉及到的 2 个 JSP 页面会被翻译成 2 个 servlet，这 2 个 servlet 的内容在执行时进行合并。
而 include 指令是静态引入，涉及到的 2 个 JSP 页面会被翻译成一个 servlet，其内容是在源文件级别进行合并。
不管是<jsp:include>标签，还是 include 指令，它们都会把两个 JSP 页面内容合并输出，所以这两个页面不要出现重复的 HTML 全局架构标签，否则输出给客户端的内容将会是一个格式混乱的 HTML 文档。
<jsp:include>标签：使用 page 属性指定被引入资源。
include 指令：使用 file 属性指定被引入资源。
假设 myweb 应用的根目录下有一个 a.jsp 文件 如果将 a.jsp 页面映射成了如下地址：

http://localhost:8080/myweb/dir1/a.html

在 a.jsp 页面中使用了如下语句引入 b.jsp 文件：

```
<jsp:include page="b.jsp" />
```

请问：b.jsp 要位于什么位置，上面的 include 才不会出错？
http://localhost:8080/myweb/b.jspf bad
http://localhost:8080/myweb/dir1/b.jspf ok
假设 myweb 应用程序的根目录下有一个 a.jsp 文件，如果将 a.jsp 页面映射为如下地址：

http://localhost:8080/myweb/dir1/a.html

在 a.jsp 页面中使用了如下语句引入 b.jspf 文件：

```
<%@ include file="b.jspf"%>
```

请问： b.jspf 要位于什么位置，上面的 include 才不会出错？
http://localhost:8080/myweb/b.jspf ok
http://localhost:8080/myweb/dir1/b.jspf bad

14.30 Tip： <jsp:forward>标签

<jsp:forward>标签用于把请求转发给另外一个资源。
语法：

```
<jsp:forward page="relativeURL | <%=expression%>" />
```

page 属性用于指定请求转发到的资源的相对路径，它也可以通过执行一个表达式来获得。

14.31 Tip： <jsp:param>标签

当使用<jsp:include>和<jsp:forward>标签引入或将请求转发给其它资源时，可以使用<jsp:param>标签向这个资源传递参数。
语法 1：

```
<jsp:include page="relativeURL | <%=expression%>">
    <jsp:param name="parameterName" value="parameterValue|<%= expression %>" />
</jsp:include>
```

语法 2：

```
<jsp:forward page="relativeURL | <%=expression%>">
    <jsp:param name="parameterName" value="parameterValue|<%= expression %>" />
</jsp:include>
```

<jsp:param>标签的 name 属性用于指定参数名，value 属性用于指定参数值。在<jsp:include>和<jsp:forward>标签中可以使用多个<jsp:param>标签来传递多个参数。

14.32 Tip：映射 JSP

```
<servlet>
    <servlet-name>SimpleJspServlet</servlet-name>
    <jsp-file>/jsp/simple.jsp</jsp-file>
    <load-on-startup>1</load-on-startup >
</servlet>
    ……
<servlet-mapping>
    <servlet-name>SimpleJspServlet</servlet-name>
    <url-pattern>/xxx/yyy.html</url-pattern>
</servlet-mapping>
```

14.33 Tip：如何查找 JSP 页面中的错误

JSP 页面中的 JSP 语法格式有问题，导致其不能被翻译成 Servlet 源文件，JSP 引擎将提示这类错误发生在 JSP 页面中的位置（行和列）以及相关信息。
JSP 页面中的 JSP 语法格式没有问题，但被翻译成的 Servlet 源文件中出现了 Java 语法问题，导致 JSP 页面翻译成的 Servlet 源文件不能通过编译，JSP 引擎也将提示这类错误发生在 JSP 页面中的位置（行和列）以及相关信息。
JSP 页面翻译成的 Servlet 程序在运行时出现异常，这与普通 Java 程序的运行时错误完全一样，Java 虚拟机将提示错误发生在 Servlet 源文件中的位置（行和列）以及相关信息。

14.34 Div 与 css

相对定位　div 没有脱离文档流 还占着原来的位置
绝对定位　position:absolute;right:0px; 相对浏览器边框定位　原来的位置让出来了

绝对定位　position:absolute;right:0px; 相对父定位　原来的位置让出来了，则父要设置为 position:relative;

14.35 JavaBean 与 Jsp

JavaBean 是一个遵循特定写法的 Java 类，它通常具有如下特点：
这个 Java 类必须具有一个无参的构造函数
属性必须私有化。
私有化的属性必须通过 public 类型的方法暴露给其它程序，并且方法的命名也必须遵守一定的命名规范。
虽然 Sun 公司在定义 JavaBean 规范时，允许 Java 开发人员把 JavaBean 设计得可以像 Swing 组件一样功能强大，但在实际的 J2EE 开发中，通常只使用到以上 JavaBean 最基本的特性。
JavaBean 在 J2EE 开发中，通常用于封装数据，对于遵循以上写法的 JavaBean 组件，其它程序可以通过反射技术实例化 JavaBean 对象，并且通过反射那些遵守命名规范的方法，从而获知 JavaBean 的属性，进而调用其属性保存数据。

14.36 Tip： JavaBean 的属性

JavaBean 的属性可以是任意类型，并且一个 JavaBean 可以有多个属性。每个属性通常都需要具有相应的 setter、getter 方法，setter 方法称为属性修改器，getter 方法称为属性访问器。
属性修改器必须以小写的 set 前缀开始，后跟属性名，且属性名的第一个字母要改为大写，例如，name 属性的修改器名称为 setName，password 属性的修改器名称为 setPassword。

属性访问器通常以小写的 get 前缀开始，后跟属性名，且属性名的第一个字母也要改为大写，例如，name 属性的访问器名称为 getName，password 属性的访问器名称为 getPassword。
一个 JavaBean 的某个属性也可以只有 set 方法或 get 方法，这样的属性通常也称之为只写、只读属性。

14.37 Tip：在 JSP 中使用 JavaBean

JSP 技术提供了三个关于 JavaBean 组件的动作元素，即 JSP 标签，它们分别为：
<jsp:useBean>标签：用于在 JSP 页面中查找或实例化一个 JavaBean 组件。
<jsp:setProperty>标签：用于在 JSP 页面中设置一个 JavaBean 组件的属性。
<jsp:getProperty>标签：用于在 JSP 页面中获取一个 JavaBean 组件的属性。

14.38 Tip： <jsp:useBean>标签

<jsp:useBean>标签用于在指定的域范围内查找指定名称的 JavaBean 对象：
如果存在则直接返回该 JavaBean 对象的引用。
如果不存在则实例化一个新的 JavaBean 对象并将它以指定的名称存储到指定的域范围中。
常用语法：

```
<jsp:useBean id="beanName" class="package.class"
            scope="page|request|session|application"/>
```

id 属性用于指定 JavaBean 实例对象的引用名称和其存储在域范围中的名称。
class 属性用于指定 JavaBean 的完整类名（即必须带有包名）。
scope 属性用于指定 JavaBean 实例对象所存储的域范围，其取值只能是 page、request、session 和 application 等四个值中的一个，其默认值是 page。

14.39 Tip：<jsp:useBean>执行原理

```
<jsp:useBean id="currentDate" class="java.util.Date"/>
```

翻译成的 Servlet 源码：

```
java.util.Date currentDate = null;
synchronized (_jspx_page_context) {
    currentDate = (java.util.Date)                    _jspx_page_context.getAttribute(
            "currentDate", PageContext.PAGE_SCOPE);
    if (currentDate == null){
        currentDate = new java.util.Date();
        _jspx_page_context.setAttribute("currentDate",
            currentDate, PageContext.PAGE_SCOPE);
    }
}
```

14.40 Tip：带标签体的<jsp:useBean>标签

语法：

```
<jsp:useBean ...>
    Body
</jsp:useBean>
```

功能：
Body 部分的内容只在<jsp:useBean>标签创建 JavaBean 的实例对象时才执行。

14.41 Tip： <jsp:setProperty>标签

<jsp:setProperty>标签用于设置和访问 JavaBean 对象的属性。
语法格式：

```
<jsp:setProperty name="beanName"
{
    property="propertyName" value="{string | <%= expression %>}" |
    property="propertyName" [ param="parameterName" ] |
    property= "*"
}/>
```

name 属性用于指定 JavaBean 对象的名称。
property 属性用于指定 JavaBean 实例对象的属性名。
value 属性用于指定 JavaBean 对象的某个属性的值，value 的值可以是字符串，也可以是表达式。为字符串时，该值会自动转化为 JavaBean 属性相应的类型，如果 value 的值是一个表达式，那么该表达式的计算结果必须与所要设置的 JavaBean 属性的类型一致。
param 属性用于将 JavaBean 实例对象的某个属性值设置为一个请求参数值，该属性值同样会自动转换成要设置的 JavaBean 属性的类型。

支持 8 种基本数据类型

```
<jsp:useBean id="person" class="cn.itcast.domain.Person" scope="page"></jsp:useBean>
<jsp:setProperty property="birthday" name="person" value="<%=new Date() %>"/>
<%=person.getBirthday() %>
```

14.42 Tip： <jsp:getProperty>标签

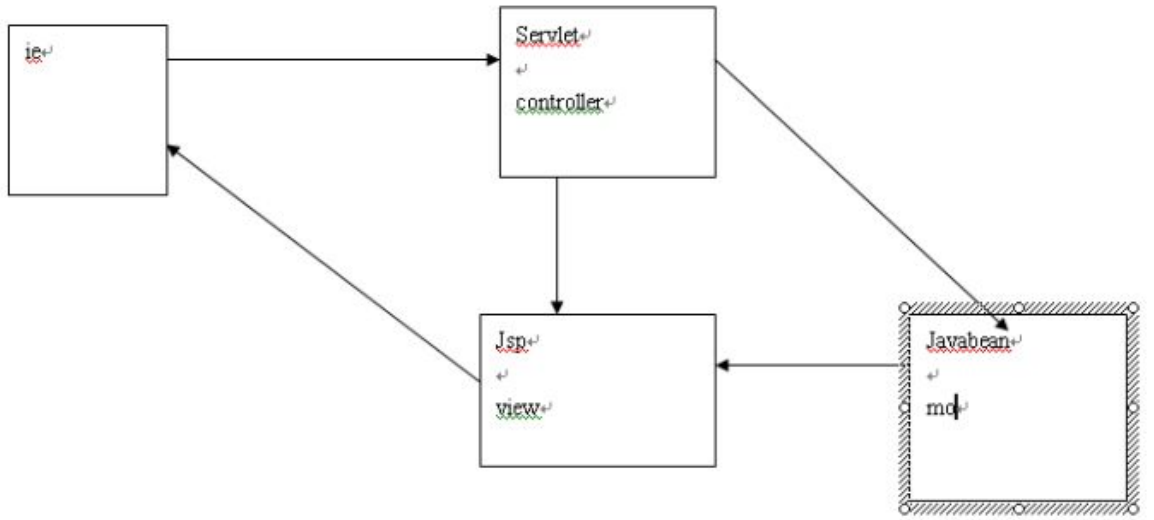
<jsp:getProperty>标签用于读取 JavaBean 对象的属性，也就是调用 JavaBean 对象的 getter 方法，然后将读取的属性值转换成字符串后插入进输出的响应正文中。
语法：

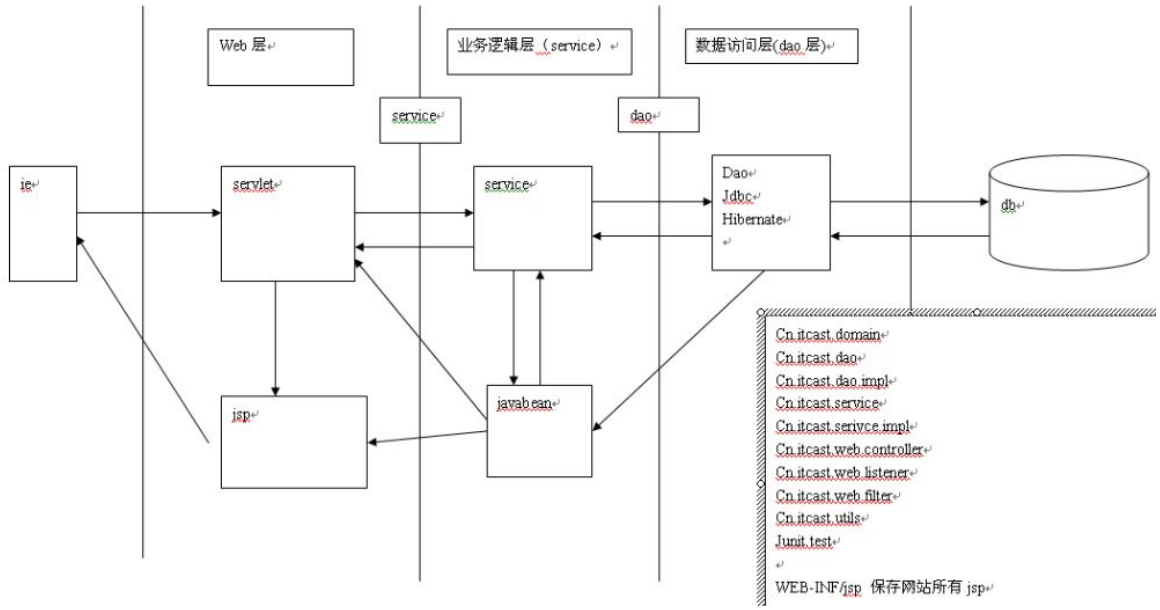
```
<jsp:getProperty name="beanInstanceName" property="PropertyName" />
```

name 属性用于指定 JavaBean 实例对象的名称，其值应与<jsp:useBean>标签的 id 属性值相同。
property 属性用于指定 JavaBean 实例对象的属性名。
如果一个 JavaBean 实例对象的某个属性的值为 null，那么，使用<jsp:getProperty>标签输出该属性的结果将是一个内容为“null”的字符串。

14.43 Tip： JSP 开发模式

SUN 公司推出 JSP 技术后，同时也推荐了两种 web 应用程序的开发模式，一种是 JSP+JavaBean 模式，一种是 Servlet+JSP+JavaBean 模式。
JSP+JavaBean 模式适合开发业务逻辑不太复杂的 web 应用程序，这种模式下，JavaBean 用于封装业务数据，JSP 即负责处理用户请求，又显示数据。
Servlet+JSP+JavaBean(MVC)模式适合开发复杂的 web 应用，在这种模式下，servlet 负责处理用户请求，jsp 负责数据显示，javabean 负责封装数据。 Servlet+JSP、JavaBean 模式程序各个模块之间层次清晰，web 开发推荐采用此种模式。






```
package cn.itcast.domain;
import java.util.Date;
//1
public class User {

	private String id;
	private String username;
	private String password;
	private String email;
	private Date birthday;
	private String nickname;
	public String getId() {
		return id;
	}
	public void setId(String id) {
		this.id = id;
	}

	public String getUsername() {
```

```
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public Date getBirthday() {
        return birthday;
    }
    public void setBirthday(Date birthday) {
        this.birthday = birthday;
    }
    public String getNickname() {
        return nickname;
    }
    public void setNickname(String nickname) {
        this.nickname = nickname;
    }
}
```

/day09_user/src/users.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<users>
    <user id="234343434" username="aaa" password="123" email="aa@sina.com"
birthday="1900-09-18" nickname="强子" />
</users>
```

```
package cn.itcast.dao;

import cn.itcast.domain.User;
//5
public interface UserDao {
    void add(User user);
    User find(String username, String password);
    //查找注册的用户是否在数据库中存在
    boolean find(String username);
}
```

```
package cn.itcast.dao.impl;

import java.text.SimpleDateFormat;

import org.dom4j.Document;
import org.dom4j.Element;

import cn.itcast.dao.UserDao;
import cn.itcast.domain.User;
import cn.itcast.utils.XmlUtils;
//2
public class UserDaoImpl implements UserDao {
	public void add(User user){
		try {
			Document document=XmlUtils.getDocument();
			Element root=document.getRootElement();
			Element user_tag=root.addElement("user");
			user_tag.setAttributeValue("id", user.getId());
			user_tag.setAttributeValue("username", user.getUsername());
			user_tag.setAttributeValue("password", user.getPassword());
			user_tag.setAttributeValue("email", user.getEmail());
			user_tag.setAttributeValue("nickname", user.getNickname());
			user_tag.setAttributeValue("birthday", user.getBirthday()==null ? "" :
user.getBirthday().toLocaleString());
			XmlUtils.write2Xml(document);
		} catch (Exception e) {
			throw new RuntimeException(e);
		}
	}
	public User find(String username,String password){

		try {
			Document document=XmlUtils.getDocument();
			Element  e=(Element)  document.selectSingleNode("//user[@username='"+username+"'
and @password='"+password+"']");
			if(e==null)
				return null;
			User user=new User();
			user.setUsername(e.attributeValue("username"));
			user.setNickname(e.attributeValue("nickname"));
			user.setPassword(e.attributeValue("password"));
			user.setEmail(e.attributeValue("email"));
			user.setId(e.attributeValue("id"));
			String date=e.attributeValue("birthday");
			if(date==null || date.equals("")){
				user.setBirthday(null);
			}else{
```

```
                SimpleDateFormat df=new SimpleDateFormat("yyyy-MM-dd");
                user.setBirthday(df.parse(date));
            }
            return user;
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
    //查找注册的用户是否在数据库中存在
    public boolean find(String username){
        try {
            Document document=XmlUtils.getDocument();
            Element e=(Element)
document.selectSingleNode("//user[@username='"+username+"']");
            if(e==null)
                return false;
            return true;
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
}
```

```
package cn.itcast.exception;
//7
public class UserExistException extends Exception {

    public UserExistException() {
        // TODO Auto-generated constructor stub
    }

    public UserExistException(String message) {
        super(message);
        // TODO Auto-generated constructor stub
    }

    public UserExistException(Throwable cause) {
        super(cause);
        // TODO Auto-generated constructor stub
    }

    public UserExistException(String message, Throwable cause) {
        super(message, cause);
        // TODO Auto-generated constructor stub
    }

/*  public UserExistException(String message, Throwable cause,
```

```
            boolean enableSuppression, boolean writableStackTrace) {
        super(message, cause, enableSuppression, writableStackTrace);
        // TODO Auto-generated constructor stub
    }*/

}
```

```
package cn.itcast.service.impl;

import cn.itcast.dao.UserDao;
import cn.itcast.dao.impl.UserDaoImpl;
import cn.itcast.domain.User;
import cn.itcast.exception.UserExistException;
import cn.itcast.utils.ServiceUtils;

//6  对 web 层提供所有的业务服务
public class BusinessServiceImpl {

    private UserDao dao=new UserDaoImpl();//工厂模式    spring
    //对 web 层提供注册服务
    public void register(User user) throws UserExistException{
        boolean b=dao.find(user.getUsername());
        if(b){
            throw new UserExistException("用户已经存在！");
            //用户存在，给 web 层抛出编译时异常
        }else{
            user.setPassword(ServiceUtils.md5(user.getPassword()));
            dao.add(user);
        }
    }
    //对 web 层提供登录服务
    public User login(String username,String password){
        password=ServiceUtils.md5(password);
        return dao.find(username, password);
    }

}
```

```
package cn.itcast.utils;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.UnsupportedEncodingException;

import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.io.OutputFormat;
```

```
import org.dom4j.io.SAXReader;
import org.dom4j.io.XMLWriter;

//3
public class XmlUtils {
	private static String filepath;
	static{
		filepath=XmlUtils.class.getClassLoader().getResource("users.xml").getPath();
		filepath=filepath.replace("%20", " ");
	}
	public static Document getDocument() throws Exception{
		SAXReader reader = new SAXReader();
		Document document = reader.read(new File(filepath));
		return document;
	}
	public static void write2Xml(Document document) throws Exception{
		OutputFormat format = OutputFormat.createPrettyPrint();
		format.setEncoding("UTF-8");
		XMLWriter writer = new XMLWriter(new FileOutputStream(filepath), format );
		writer.write( document );
		writer.close();
	}

}
```

```
package cn.itcast.utils;

import java.lang.reflect.InvocationTargetException;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Enumeration;
import java.util.UUID;

import javax.servlet.http.HttpServletRequest;

import org.apache.commons.beanutils.BeanUtils;
import org.apache.commons.beanutils.ConvertUtils;
import org.apache.commons.beanutils.Converter;

public class WebUtils {
	public static <T> T request2Bean(HttpServletRequest request,Class <T> beanClass){
		try {
			T bean = beanClass.newInstance();
			Enumeration e=request.getParameterNames();
			while(e.hasMoreElements()){
				String name=(String)e.nextElement();
```

```
                String value=request.getParameter(name);
                BeanUtils.setProperty(bean, name, value);
            }
            return bean;
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
    public static void copyBean(Object src,Object dest){
        /*
          private String username;
    private String password;
    private String password2;
    private String email;
    private String birthday;
    private String nickname;


        private String id;
    private String username;
    private String password;
    private String email;
    private Date birthday;
    private String nickname;
         * */
        ConvertUtils.register(new Converter() {
            @Override
            public Object convert(Class type, Object value) {
                if (value == null)
                    return null;
                String str = (String) value;
                if (str.trim().equals(""))
                    return null;
                SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd");
                try {
                    return df.parse(str.trim());
                } catch (ParseException e) {
                    throw new RuntimeException();
                }
            }
        }, Date.class);
        try {
            BeanUtils.copyProperties(dest, src);
        } catch (Exception e) {
            throw new RuntimeException();
        }
    }//产生全世界唯一的 id
```

```
	public static String generateID(){
		return UUID.randomUUID().toString();
	}

}
```

```
package cn.itcast.utils;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

import sun.misc.BASE64Encoder;

//8
public class ServiceUtils {

	public static String md5(String message){
		try {
			MessageDigest md=MessageDigest.getInstance("md5");
			byte md5[]=md.digest(message.getBytes());
			BASE64Encoder encoder=new BASE64Encoder();
			return encoder.encode(md5);
		} catch (NoSuchAlgorithmException e) {
			throw new RuntimeException(e);
		}
	}
}
```

```
package cn.itcast.web.formbean;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.HashMap;
import java.util.Map;

import org.apache.commons.beanutils.locale.converters.DateLocaleConverter;

public class RegisterForm {
	private String username;
	private String password;
	private String password2;
	private String email;
	private String birthday;
	private String nickname;
	private Map errors=new HashMap();

	public String getUsername() {
```

```
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public String getPassword2() {
        return password2;
    }

    public void setPassword2(String password2) {
        this.password2 = password2;
    }
    public String getEmail() {
        return email;
    }
    public void setEmail(String email) {
        this.email = email;
    }
    public String getBirthday() {
        return birthday;
    }
    public void setBirthday(String birthday) {
        this.birthday = birthday;
    }
    public String getNickname() {
        return nickname;
    }
    public void setNickname(String nickname) {
        this.nickname = nickname;
    }
    public Map getErrors() {
        return errors;
    }
    public void setErrors(Map errors) {
        this.errors = errors;
    }
    public boolean validate(){
```

```
boolean isOk=true;
//用户名不能为空，且 3-8 位字母
//密码不空，3-8 位数字
//邮箱不空，且一个合法邮箱
//生日可以为空，如果不为空，则要合法
//昵称要汉字，不能为空

if(username==null || username.trim().equals("")){
    isOk=false;
    errors.put("username","用户名不能为空");
}else{
    if(!username.matches("[A-Za-z]{3,8}")){
        isOk=false;
        errors.put("username","用户名 3-8 位字母");
    }
}
if(password==null || password.trim().equals("")){
    isOk=false;
    errors.put("password","密码不能为空");
}else{
    if(!password.matches("\\d{3,8}")){
        isOk=false;
        errors.put("password","密码 3-8 位数字");
    }
}
if(password2==null || password2.trim().equals("")){
    isOk=false;
    errors.put("password2","确认密码不能为空");
}else{
    if(!password.equals(password2)){
        isOk=false;
        errors.put("password2","两次输入密码不一样");
    }
}
if(email==null || email.trim().equals("")){
    isOk=false;
    errors.put("email","邮箱不能为空");
}else{//aa@sina.com    aa@sina.com.cn
    if(!email.matches("\\w+@\\w+(\\.\\w+)+")){
        isOk=false;
        errors.put("email","邮箱格式不合法");
    }
}
if(birthday!=null && !birthday.trim().equals("")){
    DateLocaleConverter dlc=new DateLocaleConverter();
    try{
        dlc.convert(birthday,"yyyy-MM-dd");
```

```
                }catch(Exception e){
                    isOk=false;
                    errors.put("birthday","日期格式不对");
                }
                //SimpleDateFormat df=new SimpleDateFormat("yyyy-MM-dd");
            }
            if(nickname==null || nickname.trim().equals("")){
                isOk=false;
                errors.put("nickname","昵称不能为空");
            }else{//汉字区间   \u4e00-\u9fa5
                if(!nickname.matches("^([\u4e00-\u9fa5]+)$")){
                    isOk=false;
                    errors.put("nickname","昵称必须为汉字");
                }
            }
            return isOk;
        }
}
```

```
package cn.itcast.web.controller;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import cn.itcast.domain.User;
import cn.itcast.exception.UserExistException;
import cn.itcast.service.impl.BusinessServiceImpl;
import cn.itcast.utils.WebUtils;
import cn.itcast.web.formbean.RegisterForm;
//处理注册请求
public class RegisterServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        request.setCharacterEncoding("UTF-8");
        //1.合法性校验   formBean 把表单数据封装到 formbean
        RegisterForm form=WebUtils.request2Bean(request,RegisterForm.class);
        boolean b=form.validate();
        //2.如果校验失败，跳回到表单页面，回显失败消息
        if(!b){
            request.setAttribute("form", form);
```

```
                request.getRequestDispatcher("/WEB-INF/jsp/register.jsp").forward(request, response);
                return ;
            }
            //3.校验成功，则调用 service 处理注册请求
            BusinessServiceImpl service =new BusinessServiceImpl();

            User user=new User();
            WebUtils.copyBean(form, user);
            user.setId(WebUtils.generateID());
            try {
                service.register(user);
                //6.service 处理成功，跳转到网站的全局消息显示页面，为用户显示注册成功消息
                request.setAttribute("message","注册成功");
                request.getRequestDispatcher("/message.jsp").forward(request, response);
                return;
            } catch (UserExistException e) {
                //4.service 处理不成功，原因是用户存在跳回注册页面显示注册用户已存在
                form.getErrors().put("username", "注册的用户已存在");
                request.setAttribute("form", form);
                request.getRequestDispatcher("/WEB-INF/jsp/register.jsp").forward(request, response);
                return;
            }catch (Exception e) {
//5.service 处理不成功，原因是其他问题    ，跳转到全局消息处理界面，显示友好错误页面
                e.printStackTrace();
                request.setAttribute("message", "服务器出现未知错误");
                request.getRequestDispatcher("/message.jsp").forward(request, response);
                return;
            }
        }

        public void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {

            doGet(request,response);
        }
}
```

```
package cn.itcast.web.controller;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
import cn.itcast.domain.User;
import cn.itcast.service.impl.BusinessServiceImpl;

public class LoginServlet extends HttpServlet {

	public void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {

		String username=request.getParameter("username");
		String password=request.getParameter("password");
		BusinessServiceImpl service	=new BusinessServiceImpl();
		User user=service.login(username, password);
		if(user!=null){
			request.getSession().setAttribute("user", user);
			response.sendRedirect(request.getContextPath()+"/index.jsp");
			return ;
		}
		request.setAttribute("message", "用户名或密码错误");
		request.getRequestDispatcher("/message.jsp").forward(request, response);
	}

	public void doPost(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		doGet(request,response);
	}
}
```

```
package cn.itcast.web.controller;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class LoginOutServlet extends HttpServlet {

	public void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		HttpSession session=request.getSession(false);
		if(session!=null)
			session.removeAttribute("username");
		request.getSession().setAttribute("message", "注销成功!3 秒后跳转 <br/><meta
http-equiv='refresh' content='3;"+request.getContextPath()+"/'>");
```

```
            response.sendRedirect(request.getContextPath()+"/message.jsp");
        }

        public void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
        }

}
```

```
package cn.itcast.web.UI;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
//提供登陆界面
public class LoginUIServlet extends HttpServlet {

        public void doGet(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
            request.getRequestDispatcher("/WEB-INF/jsp/login.jsp").forward(request, response);
        }
        public void doPost(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
            doGet(request,response);
        }
}
```

```
package cn.itcast.web.UI;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class RegisterUIServlet extends HttpServlet {
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException {
            request.getRequestDispatcher("/WEB-INF/jsp/register.jsp").forward(request, response);
        }
        public void doPost(HttpServletRequest req, HttpServletResponse resp)
                throws ServletException, IOException {
```

```
            doGet(req,resp);
        }
}
```

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>全局消息显示页面  message.jsp</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
  </head>
  <body>
    ${message }<br>
  </body>
</html>
```

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>index.jsp</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
  </head>
  <body>
  <c:if test="${user!=null }">
  欢迎：${user.username } <a href="${pageContext.request.contextPath }/servlet/loginOutServlet">注销</a><br/>
  </c:if>
   <a href="${pageContext.request.contextPath }/servlet/registerUIServlet">注册</a> <br>
   <a href="${pageContext.request.contextPath }/servlet/loginUIServlet">登录</a> <br>
    </body>
</html>
```

1.搭建开发环境

1.1 导开发包

dom4j http://dom4j.sourceforge.net/

dom4j\lib\jaxen-1.1-beta--.jar

jstl

beanUtils

log4j

1.2 创建组织程序的包

cn.itcast.domain

cn.itcast.dao

cn.itcast.dao.impl
cn.itcast.service
cn.itcast.service.impl
cn.itcast.web.controller 处理请求的 servlet
cn.itcast.web.UI 给用户提供界面
cn.itcast.utils
junit.test

/WEB-INF/jsp 保存网站所以 jsp

1.3 创建代表数据库的 xml 文件

在类目录下创建一个代表数据库的 user.xml

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>登陆界面 WEB-INF/jsp/login.jsp</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
  </head>

  <body>
  <form action="${pageContext.request.contextPath }/servlet/loginServlet" method="post">
    用户名<input type="text" name="username"/><br/>
    密码<input type="text" name="password"/><br/>
    <input type="submit" value="登陆"/><br/>
  </form>

  </body>
</html>
```

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>注册页面 WEB-INF/jsp/register.jsp</title>
    <meta http-equiv="pragma" content="no-cache">
    <meta http-equiv="cache-control" content="no-cache">
    <meta http-equiv="expires" content="0">
  </head>
  <body> <form method="post" action="${pageContext.request.contextPath }/servlet/registerServlet"
name="reg">
    <table width="320" border="1" height="172">
<tbody><tr>
```

```
<td> 姓名</td>
<td> <input type="text" name="username"
value="${form.username }"><span>${form.errors.username }</span></td></tr>
<tr>
<td> 密码</td>
<td> <input type="password"
name="password"><span>${form.errors.password }</span></td></tr>
<tr>
<td> 确认密码</td>
<td> <input type="password"
name="password2"><span>${form.errors.password2}</span></td></tr>
<tr>
<td> email</td>
<td> <input type="text" name="email"
value="${form.email }"><span>${form.errors.email }</span></td></tr>
<tr>
<td> 生日</td>
<td> <input type="text" name="birthday"
value="${form.birthday }"><span>${form.errors.birthday }</span></td></tr>
<tr>
<td> 别名</td>
<td> <input type="text" name="nickname"
value="${form.nickname}"><span>${form.errors.nickname }</span></td></tr>
<tr>
<td> <input type="reset" value="重置" name="reset"></td>
<td> <input type="submit" value="提交" name="submit"></td></tr>
</tbody></table>
</form>
  </body>
</html>
```

1.搭建开发环境

1.1 导入开发包

jstl

1.2 创建组织程序的包

cn.itcast.domain

cn.itcast.dao

cn.itcast.service

cn.itcast.web.controller

cn.itcast.web.UI

cn.itcast.utils

WEB-INF/jsp

1.3 创建代表数据库的 DB

2.开发 Dao

3.开发 service

4.开发 web

Index.jsp

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
  </head>
  <body>
    <a href="${pageContext.request.contextPath }/servlet/ListBookServlet">浏览书籍</a><br>
  </body>
</html>
```

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core"      prefix="c"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>书籍列表页面  WEB-INF/jsp/listbook.jsp</title>
  </head>

  <body style="text-align:center">
  <h1>书籍列表</h1>
    <table width="70%" border="1">
        <tr>
            <td>书名</td>
            <td>作者</td>
            <td>售价</td>
            <td>描述</td>
            <td>操作</td>
        </tr>
        <c:forEach var="entry" items="${map}">
        <tr>
            <td>${entry.value.name }</td>
            <td>${entry.value.author }</td>
            <td>${entry.value.price }</td>
            <td>${entry.value.description }</td>
            <td><a    href="${pageContext.request.contextPath    }/servlet/BuyServlet?id=${entry.key}"
target="_blank">购买</a></td>



        </tr>
        </c:forEach>
```

```
    </table>
  </body>
</html>
```

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core"      prefix="c"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>购物车列表  WEB-INF/jsp/listcart.jsp</title>
    <script type="text/javascript">
        function deleteitem(id){
            var b=window.confirm("真的删除吗?");
            if(b){

    window.location="${pageContext.request.contextPath }/servlet/DeleteItemServlet?id="+id;
            }
        }
        function clearcart(){
            var b=window.confirm("真的清空吗?");
            if(b){
                window.location="${pageContext.request.contextPath }/servlet/ClearCartServlet";
            }
        }
        function changeQuantity(input,id,oldvalue){
            //alert(input.value);
            //检查数量是不是一个数字，且是正整数
            var quantity=input.value;
            if(isNaN(quantity)){
                alert("请输入数字");
                input.value=oldvalue;
                return;
            }
            if(quantity<0 || quantity!=parseInt(quantity)){
                alert("请输入正整数");
                input.value=oldvalue;
                return;
            }
            var b=window.confirm("确定修改吗?");
            if(b){

    window.location="${pageContext.request.contextPath   }/servlet/ChangeQuantityServlet?id="+id+"&q
uantity="+quantity;
            }
        }
    </script>
```

```
  </head>

  <body style="text-align:center">

  <h1>购物车列表</h1>
  <c:if test="${empty(cart.map)}">
   你没有购买任何商品
  </c:if>
    <c:if test="${!empty(cart.map)}">
    <table width="70%" border="1">
        <tr>
            <td>书名</td>
            <td>作者</td>
            <td>单价</td>
            <td>数量</td>
            <td>小计</td>
            <td>操作</td>
        </tr>
        <c:forEach var="entry" items="${cart.map}">
        <tr>
            <td>${entry.value.book.name }</td>
            <td>${entry.value.book.author }</td>
            <td>${entry.value.book.price }</td>
            <td>
                <input type="text" name="quantity" value="${entry.value.quantity }"
onchange="changeQuantity(this,${entry.key},${entry.value.quantity })"/>
            </td>
            <td>${entry.value.price }</td>
            <td><a href="javascript:void(0);" target="_blank" onclick="deleteitem(${entry.key});return
false;">删除</a></td>

        </tr>
        </c:forEach>
        <tr>
            <td colspan="3">总价计</td>
            <td colspan="2">${cart.price }元</td>
            <td colspan="1"><a href="javascript:void(0);" target="_blank"
onclick="clearcart();return false;">清空购物车</a></td>
        </tr>

    </table>
    </c:if>
  </body>
</html>
```

```
package cn.itcast.DB;
```

```
import java.util.LinkedHashMap;
import java.util.Map;

import cn.itcast.domain.Book;

public class DB {
	private static Map map=new LinkedHashMap();
	static{
		map.put("1", new Book("1","javaweb 开发","老张",38,"一本好书"));
		map.put("2", new Book("2","jdbc 开发","老黎",18,"一本好书"));
		map.put("3", new Book("3","ajax 开发","老佟",328,"一本好书"));
		map.put("4", new Book("4","jbpm 开发","老毕",58,"一本好书"));
		map.put("5", new Book("5","struts 开发","老方",28,"一本好书"));
		map.put("6", new Book("6","spring 开发","老方",98,"一本好书"));
	}
	public static Map getAll(){
		return map;
	}
}
```

```
package cn.itcast.domain;

public class Book {
	private String id;
	private String name;
	private String author;
	private double price;
	public Book() {
		super();
	}
	public Book(String id, String name, String author, double price,
			String description) {
		super();
		this.id = id;
		this.name = name;
		this.author = author;
		this.price = price;
		this.description = description;
	}
	public String getId() {
		return id;
	}
	public void setId(String id) {
		this.id = id;
	}
	public String getName() {
		return name;
```

```
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getAuthor() {
		return author;
	}
	public void setAuthor(String author) {
		this.author = author;
	}
	public double getPrice() {
		return price;
	}
	public void setPrice(double price) {
		this.price = price;
	}
	public String getDescription() {
		return description;
	}
	public void setDescription(String description) {
		this.description = description;
	}
	private String description;
}
```

```
package cn.itcast.domain;

import java.util.Map;

import cn.itcast.DB.DB;

public class BookDao {
	public Map getAll(){
		return DB.getAll();
	}
	public Book find(String id){
		return (Book) DB.getAll().get(id);
	}
}
```

```
package cn.itcast.domain;

import java.util.LinkedHashMap;
import java.util.Map;

//代表用户的购物车
public class Cart {
	private Map<String,CartItem> map=new LinkedHashMap();
```

```
    private double price;//所有商品多少前
    public void add(Book book){
        CartItem item=map.get(book.getId());
        if(item==null){
            item=new CartItem();
            item.setBook(book);
            item.setQuantity(1);
            map.put(book.getId(), item);
        }else{
            item.setQuantity(item.getQuantity()+1);
        }
    }
    public Map<String, CartItem> getMap() {
        return map;
    }
    public void setMap(Map<String, CartItem> map) {
        this.map = map;
    }
    public double getPrice() {
        double totalprice=0;
        for(Map.Entry<String, CartItem> entry:map.entrySet()){
            CartItem item=entry.getValue();
            totalprice+=item.getPrice();
        }
        this.price=totalprice;
        return price;
    }
    public void setPrice(double price) {
        this.price = price;
    }
}
```

```
package cn.itcast.domain;
//代表某个商品，以及商品出现的次数（购物项）
public class CartItem {
    private Book book;
    private int quantity;
    private double price;
    public Book getBook() {
        return book;
    }
    public void setBook(Book book) {
        this.book = book;
    }
    public int getQuantity() {
        price=book.getPrice()*quantity;
        return quantity;
```

```
	}
	public void setQuantity(int quantity) {
		this.quantity = quantity;
	}
	public double getPrice() {
		return price;
	}
	public void setPrice(double price) {
		this.price = price;
	}
}
```

```
package cn.itcast.service;

import java.util.Map;

import cn.itcast.domain.Book;
import cn.itcast.domain.BookDao;
import cn.itcast.domain.Cart;
import cn.itcast.domain.CartItem;
//业务层，统一对 web 层提供所有服务
public class BusinessService {
	private BookDao dao=new BookDao();
	public Map getAll(){
		return dao.getAll();
	}
	public Book findBook(String id){
		return (Book) dao.getAll().get(id);
	}
	//删除购物车中的购物项
	public void deleteCartItem(String id, Cart cart) {
		cart.getMap().remove(id);
	}
	public void clearCart(Cart cart) {
		cart.getMap().clear();
	}
	public void changeItemQuantity(String id, String quantity, Cart cart) {
		CartItem item=cart.getMap().get(id);
		item.setQuantity(Integer.parseInt(quantity));
	}
}
```

```
package cn.itcast.web.controller;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
```

```
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import cn.itcast.domain.Book;
import cn.itcast.domain.Cart;
import cn.itcast.service.BusinessService;

public class BuyServlet extends HttpServlet {

	public void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		String id=request.getParameter("id");
		BusinessService service = new BusinessService();
		Book book=service.findBook(id);
		//得到用户的购物车
		Cart cart=(Cart) request.getSession().getAttribute("cart");
		if(cart==null){
			cart=new Cart();
			request.getSession().setAttribute("cart", cart);
		}
		//把书加到用户的购物车中
		cart.add(book);
		response.sendRedirect(request.getContextPath()+"/servlet/ListcartUIServlet");
	}

	public void doPost(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
	}

}
```

```
package cn.itcast.web.controller;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import cn.itcast.domain.Cart;
import cn.itcast.service.BusinessService;
//把购物车中的书修改为指定数量
public class ChangeQuantityServlet extends HttpServlet {
	public void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
            throws ServletException, IOException {
        String id=request.getParameter("id");
        String quantity=request.getParameter("quantity");
        Cart cart=(Cart)request.getSession().getAttribute("cart");
        BusinessService service=new BusinessService();
        service.changeItemQuantity(id,quantity,cart);
        request.getRequestDispatcher("/WEB-INF/jsp/listcart.jsp").forward(request, response);
    }
}
```

```
package cn.itcast.web.controller;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import cn.itcast.domain.Cart;
import cn.itcast.service.BusinessService;
//清空购物车
public class ClearCartServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        Cart cart=(Cart) request.getSession().getAttribute("cart");
        BusinessService service = new BusinessService();
        service.clearCart(cart);
        request.getRequestDispatcher("/WEB-INF/jsp/listcart.jsp").forward(request, response);
    }
}
```

```
package cn.itcast.web.controller;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import cn.itcast.domain.Cart;
import cn.itcast.service.BusinessService;

public class DeleteItemServlet extends HttpServlet {
```

```
	public void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		String id=request.getParameter("id");
		BusinessService service=new BusinessService();
		Cart cart=(Cart) request.getSession().getAttribute("cart");
		service.deleteCartItem(id,cart);
		request.getRequestDispatcher("/WEB-INF/jsp/listcart.jsp").forward(request, response);
	}
}
```

```
package cn.itcast.web.controller;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Map;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import cn.itcast.service.BusinessService;

public class ListBookServlet extends HttpServlet {

	public void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		BusinessService service =new BusinessService();
		Map map=service.getAll();
		request.setAttribute("map", map);
		request.getRequestDispatcher("/WEB-INF/jsp/listbook.jsp").forward(request, response);
	}

	public void doPost(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		doGet(request,response);
	}
}
```

```
package cn.itcast.web.UI;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```
import javax.servlet.http.HttpServletResponse;

public class ListcartUIServlet extends HttpServlet {
	public void doGet(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		request.getRequestDispatcher("/WEB-INF/jsp/listcart.jsp").forward(request, response);
	}
}
```

14.44 Tip： EL 表达式和 JSTL 标签快速入门

<%
String data="my data";
request.setAttribute("data",data);
%>
${data } <%--pageContext.findAttribute("data")
page request session application --%>
	<%
	Person p=new Person();
	p.setName("namenamename");
	request.setAttribute("p",p);
	%>
	${p.name }

<%
Person p1=new Person();
Address a=new Address();
a.setCity("huang gang");
p1.setAddress(a);
request.setAttribute("p1",p1);
%>
${p1.address.city }
	<%
	List list=new ArrayList();
	list.add(new Address("上海"));
	list.add(new Address("北京 2"));
	list.add(new Address("武汉"));
	request.setAttribute("list",list);
	%>
	${list[1].city } <%--北京 2 --%>
	<c:forEach var="a" items="${list}">
	${a.city}
	</c:forEach>

<% Map map=new HashMap(); map.put("aa",new Address("上海")); map.put("bb",new Address("北京 2")); map.put("cc",new Address("武汉")); map.put("111",new Address("南京")); request.setAttribute("map",map); %> ${map.aa.city } <%--上海 --%> ${map["111"].city } <%--南京 ,通常用点号， 点号取不出来用[]--%> <c:forEach var="b" items="${map }"> ${b.key } : ${b.value.city } </c:forEach>	${pageContext.request.contextPath }<%--/day09 --%> <c:if test="${user!=null }"> 欢迎 </c:if> <c:if test="${user==null }"> 请登录 </c:if>
http://jstl.java.net/	

EL 表达式用于获取数据，在 JSP 页面中可使用${标识符}的形式，通知 JSP 引擎调用 pageContext.findAttribute()方法，以标识符为关键字从各个域对象中获取对象。如果域对象中不存在标识符所对应的对象，则返回结果为""（注意，不是 null）。
示例：使用ＥＬ表达式获取 request、session、applection 域中的数据。
EL 表达式中也可以使用${customerBean.address}的形式来访问 JavaBean 对象的属性。
示例：使用ＥＬ表达式获取 Bean 属性。
结合 JSTL 标签，EL 表达式也可轻松获取各种集合中的元素。
示例：使用ＥＬ表达式获取 List、Map 集合中的元素。
EL 表达式也可使用类如${1==1}的形式进行简单的逻辑判断。
JSTL 标签库
JSTL 是 sun 公司开发的一套标签库，使用 JSTL 可以在页面中实现一些简单的逻辑，从而替换页面中的脚本代码。
在页面中使用 JSTL 标签需完成以下 2 个步骤：
http://tomcat.apache.org/taglibs/standard/
在 Referenced Libraries/standard.jap/META-INF/c.tld 有 uri 定义
C:\Tomcat 7.0\webapps\examples\WEB-INF\lib　　有该文件
１、导入 jstl.jar 和 standerd.jar 这两个 JSTL 的 jar 文件。
２、在 JSP 页面中使用<%@ tablib uri="" 　prifix="" %>元素导入标签库。
JSTL 标签库中常用标签：
<c:foreach var="" items="">
<c:if test="">

右键/重构/move 可把一个类移动到隐藏的父包中，

15　自定义标签库开发

自定义标签主要用于移除 Jsp 页面中的 java 代码。

移除 jsp 页面中的 java 代码，只需要完成两个步骤：
编写一个实现 Tag 接口的 Java 类，并覆盖 doStartTag 方法，把 jsp 页面中的 java 代码写到 doStartTag 方法中。
编写标签库描述符（tld）文件，在 tld 文件中对自定义标签进行描述。
完成以上操作，即可在 JSP 页面中导入和使用自定义标签。

15.1　快速入门：使用自定义标签输出客户机 IP

```
package cn.itcast.web.tag;

import java.io.IOException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;

public class ViewIPTag extends TagSupport {
```

```
    @Override
    public int doStartTag() throws JspException {
        HttpServletRequest request=(HttpServletRequest) this.pageContext.getRequest();
        String strIP=request.getRemoteAddr();
        JspWriter out=this.pageContext.getOut();
        try {
            out.write("你的 IP 地址为"+strIP);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        return super.doStartTag();
    }
}
```

```
//Itcast.tld
<?xml version="1.0" encoding="UTF-8" ?>

<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
    version="2.0">
    <description>A tag library exercising SimpleTag handlers.</description>
    <tlib-version>1.0</tlib-version>
    <short-name>SimpleTagLibrary</short-name>
    <uri>http://www.hgnc.net/jsp2-tag</uri>
    <tag>
        <description>show client IP</description>
        <name>viewIP</name>
        <tag-class>cn.itcast.web.tag.ViewIPTag</tag-class>
        <body-content>empty</body-content>
    </tag>
</taglib>
```

```
//web.xml    tld 文件放在 WEB-INF 目录下，可不配置,可参看 tomcat 例子
    <jsp-config>
        <taglib>
            <taglib-uri>
                http://www.hgnc.net/jsp2-tag
            </taglib-uri>
            <taglib-location>
                /WEB-INF/itcast.tld
            </taglib-location>
        </taglib>
    </jsp-config>
```

```
<%@ page language="java"    pageEncoding="UTF-8"%>
<%@ taglib prefix="itcast" uri="http://www.hgnc.net/jsp2-tag"%>
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>显示 IP 地址</title>
  </head>
  <body>
        <%
    String strIP=request.getRemoteAddr();
    out.print("你的 IP 地址为"+strIP);
     %> <br>

     < itcast:viewIP/>
  </body>
</html>
```

查看 tag 接口 api 文档，分析自定义标签的执行流程。

15.2 Tip：Tag 接口的执行流程

JSP 引擎将遇到自定义标签时，首先创建标签处理器类的实例对象，然后按照 JSP 规范定义的通信规则依次调用它的方法。

1、public void setPageContext(PageContext pc)， JSP 引擎实例化标签处理器后，将调用 setPageContext 方法将 JSP 页面的 pageContext 对象传递给标签处理器，标签处理器以后可以通过这个 pageContext 对象与 JSP 页面进行通信。

2、public void setParent(Tag t)，setPageContext 方法执行完后，WEB 容器接着调用的 setParent 方法将当前标签的父标签传递给当前标签处理器，如果当前标签没有父标签，则传递给 setParent 方法的参数值为 null。

3、public int doStartTag()，调用了 setPageContext 方法和 setParent 方法之后，WEB 容器执行到自定义标签的开始标记时，就会调用标签处理器的 doStartTag 方法。

4、public int doEndTag()，WEB 容器执行完自定义标签的标签体后，就会接着去执行自定义标签的结束标记，此时，WEB 容器会去调用标签处理器的 doEndTag 方法。

5、public void release()，通常 WEB 容器执行完自定义标签后，标签处理器会驻留在内存中，为其它请求服务器，直至停止 web 应用时，web 容器才会调用 release 方法。

15.3 Tip：自定义标签功能扩展

自定义标签除了可以移除 Jsp 页面中的 java 代码外，它还可以用于完成一些页面逻辑，例如：

```
    public int doStartTag() throws JspException {
        return this.SKIP_BODY;
    }
```

<tag> <description>忽略标签体</description> <name>skipBody</name> <tag-class>cn.itcast.web.tag.TagDemo1</tag-class> <body-content>JSP</body-content> </tag>	//通过自定义标签可以控制 jsp 页面某一部分内容是否执行 <itcast:skipBody> sssssssssssss </itcast:skipBody>

例如：<c:if>标签

通过自定义标签可以控制标签后的 jsp 页面是否执行。
public int doEndTag() throws JspException {
return SKIP_PAGE;
}
 	
<tag>
<description>整个 jsp 页面标签后的内容是否输出</description>
<name>skipPage</name>
 	
<tag-class>cn.itcast.web.tag.TagDemo2</tag-class>
<body-content>JSP</body-content>
</tag>
	<%@ page language="java" pageEncoding="UTF-8"%>
	<%@ taglib prefix="itcast" uri="http://www.hgnc.net/jsp2-tag"%>
	
	<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
	<html>
	<head>
	<title>整个 jsp 是否输出</title>
	</head>
	uuu
	<itcast:skipPage>kkkkkkkkkkkkl</itcast:skipPage>
	<body>
	gggg
	</body>
	</html>

15.4 通过自定义标签可以控制 jsp 页面某一部分内容重复执行。

```
int x;
@Override
public int doAfterBody() throws JspException {
    if(x-->0)
        return IterationTag.EVAL_BODY_AGAIN;
    else
        return IterationTag.SKIP_BODY;
}

@Override
public int doStartTag() throws JspException {
    x=5;
    return this.EVAL_BODY_INCLUDE;
}
```

```
<tag>
    <description>循环输出标签体</description>
    <name>loopbody</name>

<tag-class>cn.itcast.web.tag.TagDemo4</tag-class>

<body-content>JSP</body-content>
</tag>
```

例如：<c:foreach>标签

15.5 通过自定义标签可以修改 jsp 页面内容输出。

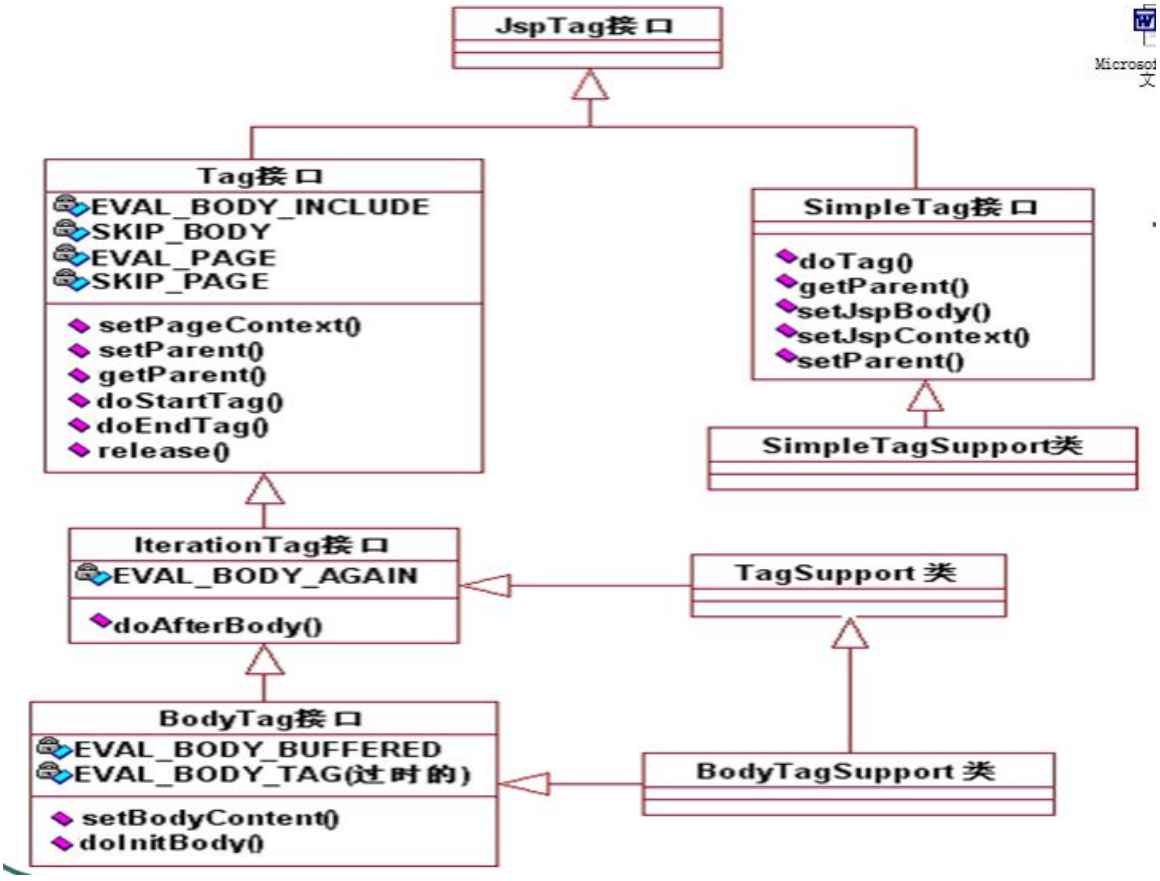
```
@Override
public int doStartTag() throws JspException {
    return BodyTag.EVAL_BODY_BUFFERED;
}
```

```
<itcast:modibody>
        kkkkkkkkkkkkl<br />
</itcast:modibody>
```

```java
@Override
public int doEndTag() throws JspException {
    BodyContent bc=this.getBodyContent();
    String content=bc.getString();
    content=content.toUpperCase();
    try {

    this.pageContext.getOut().write(content);
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return super.doEndTag();
}
```

tld 文件中的四种标签体类型

EMPTY　JSP　scriptless　tagdepentend



16　Tip：　简单标签

由于传统标签使用三个标签接口来完成不同的功能，显得过于繁琐，不利于标签技术的推广， SUN 公司为降低标签技术的学习难度，在 JSP 2.0 中定义了一个更为简单、便于编写和调用的 SimpleTag 接口来实现标签的功能。实

现 SimpleTag 接口的标签通常称为简单标签。简单标签共定义了 5 个方法：
setJspContext 方法
setParent 和 getParent 方法
setJspBody 方法
doTag 方法

16.1 Tip：SimpleTag 方法介绍(课后阅读 API)

setJspContext 方法
用于把 JSP 页面的 pageContext 对象传递给标签处理器对象
setParent 方法
用于把父标签处理器对象传递给当前标签处理器对象
getParent 方法
用于获得当前标签的父标签处理器对象
setJspBody 方法
用于把代表标签体的 JspFragment 对象传递给标签处理器对象
doTag 方法
用于完成所有的标签逻辑，包括输出、迭代、修改标签体内容等。在 doTag 方法中可以抛出 javax.servlet.jsp.SkipPageException 异常，用于通知 WEB 容器不再执行 JSP 页面中位于结束标记后面的内容，这等效于在传统标签的 doEndTag 方法中返回 Tag.SKIP_PAGE 常量的情况。

16.2 Tip：SimpleTag 接口方法的执行顺序

当 web 容器开始执行标签时，会调用如下方法完成标签的初始化
WEB 容器调用标签处理器对象的 setJspContext 方法，将代表 JSP 页面的 pageContext 对象传递给标签处理器对象。
WEB 容器调用标签处理器对象的 setParent 方法，将父标签处理器对象传递给这个标签处理器对象。注意，只有在标签存在父标签的情况下，WEB 容器才会调用这个方法。
如果调用标签时设置了属性，容器将调用每个属性对应的 setter 方法把属性值传递给标签处理器对象。如果标签的属性值是 EL 表达式或脚本表达式，则 WEB 容器首先计算表达式的值，然后把值传递给标签处理器对象。
如果简单标签有标签体，容器将调用 setJspBody 方法把代表标签体的 JspFragment 对象传递进来。
执行标签时：
容器调用标签处理器的 doTag()方法，开发人员在方法体内通过操作 JspFragment 对象，就可以实现是否执行、迭代、修改标签体的目的。

16.3 Tip：JspFragment 类

javax.servlet.jsp.tagext.JspFragment 类是在 JSP2.0 中定义的，它的实例对象代表 JSP 页面中的一段符合 JSP 语法规范的 JSP 片段，这段 JSP 片段中不能包含 JSP 脚本元素。

WEB 容器在处理简单标签的标签体时，会把标签体内容用一个 JspFragment 对象表示，并调用标签处理器对象的 setJspBody 方法把 JspFragment 对象传递给标签处理器对象。JspFragment 类中只定义了两个方法，如下所示：

getJspContext 方法
用于返回代表调用页面的 JspContext 对象.

public abstract void invoke(java.io.Writer out)
用于执行 JspFragment 对象所代表的 JSP 代码片段
参数 out 用于指定将 JspFragment 对象的执行结果写入到哪个输出流对象中，如果传递给参数 out 的值为 null，则将执行结果写入到 JspContext.getOut()方法返回的输出流对象中。(简而言之，可以理解为写给浏览器)

显示标签体

```
public class SimpleDemo1 extends SimpleTagSupport {
    @Override
    public void doTag() throws JspException, IOException {
        JspFragment jf=this.getJspBody();
        jf.invoke(this.getJspContext().getOut());
        super.doTag();
    }
}
```

不显示标签体

```
public class SimpleDemo1 extends SimpleTagSupport {
    @Override
    public void doTag() throws JspException, IOException {

    }
}
```

```
//重复显示
public class SimpleDemo2 extends SimpleTagSupport {
    @Override
    public void doTag() throws JspException, IOException {
        JspFragment jf=this.getJspBody();
        for(int i=0;i<5;i++)
            jf.invoke(this.getJspContext().getOut());
            //与这个等价jf.invoke(null);
        super.doTag();
    }
}
```

```
//修改标签内容
public class SimpleDemo3 extends SimpleTagSupport {
    @Override
    public void doTag() throws JspException, IOException {
        JspFragment jf=this.getJspBody();
        StringWriter sw=new StringWriter();
        jf.invoke(sw);

        String content=sw.toString();
        content=content.toUpperCase();

        this.getJspContext().getOut().write(content);
    }
}
```

```
//停止显示标签后面文档内容
public class SimpleDemo4 extends SimpleTagSupport {
    @Override
    public void doTag() throws JspException, IOException {
        throw new SkipPageException();
```

```
    }
}
```

```
    <uri>http://www.hgnc.net/jsp2-tagSimple</uri>
    <tag>
        <name>showbody</name>
        <tag-class>cn.itcast.web.simpletag.SimpleDemo1</tag-class>
        <body-content>scriptless</body-content>
    </tag>
```

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@ taglib prefix="itcast" uri="http://www.hgnc.net/jsp2-tagSimple"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>简单标签</title>
    </head>

    <body>
        <itcast:showbody>sssssssssssss
        </itcast:showbody>
    </body>
</html>
```

传统标签执行完后，驻留在内存象 servlet 一样，简单标签，执行完一次后，就释放了。

16.4 Tip：invoke 方法详解

JspFragment.invoke 方法可以说是 JspFragment 最重要的方法，利用这个方法可以控制是否执行和输出标签体的内容、是否迭代执行标签体的内容或对标签体的执行结果进行修改后再输出。例如：
在标签处理器中如果没有调用 JspFragment.invoke 方法，其结果就相当于忽略标签体内容；
在标签处理器中重复调用 JspFragment.invoke 方法，则标签体内容将会被重复执行；
若想在标签处理器中修改标签体内容，只需在调用 invoke 方法时指定一个可取出结果数据的输出流对象（例如 StringWriter），让标签体的执行结果输出到该输出流对象中，然后从该输出流对象中取出数据进行修改后再输出到目标设备，即可达到修改标签体的目的。

16.5 Tip：开发带属性的标签

自定义标签可以定义一个或多个属性，这样，在 JSP 页面中应用自定义标签时就可以设置这些属性的值，通过这些属性为标签处理器传递参数信息，从而提高标签的灵活性和复用性。
要想让一个自定义标签具有属性，通常需要完成两个任务：
在标签处理器中编写每个属性对应的 setter 方法
在 TLD 文件中描术标签的属性

为自定义标签定义属性时，每个属性都必须按照 JavaBean 的属性命名方式，在标签处理器中定义属性名对应的 setter 方法，用来接收 JSP 页面调用自定义标签时传递进来的属性值。 例如属性 url，在标签处理器类中就要定义相应的 setUrl(String url)方法。

在标签处理器中定义相应的 set 方法后，JSP 引擎在解析执行开始标签前，也就是调用 doStartTag 方法前，会调用 set 属性方法，为标签设置属性。

16.6 Tip：在 TLD 中描述标签属性

元素名	是否必须指定	描　　述
description	否	用于指定属性的描述信息。
name	是	用于指定属性的名称。属性名称是大小写敏感的，并且不能以 jsp、_jsp、java 和 sun 开头。
required	否	用于指定在 JSP 页面中调用自定义标签时是否必须设置这个属性。其取值包括 true 和 false，默认值为 false，true 表示必须设置，否则可以设置也可以不设置该属性。
rtexprvalue	否	rtexprvalue 是 runtime expression value（运行时表达式）的英文简写，用于指定属性值是一个静态值或动态值。其取值包括 true 和 false，默认值为 false，false 表示只能为该属性指定静态文本值，例如"123"；true 表示可以为该属性指定一个 JSP 动态元素，动态元素的结果作为属性值，例如 JSP 表达式<%=value %>。
type	否	用于指定属性值的 Java 类型。

16.7

```
<%@ page language="java" pageEncoding="UTF-8"%>
<%@page import="java.util.Date"%>
<%@ taglib prefix="itcast" uri="http://www.hgnc.net/jsp2-tagSimple"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <title>带属性标签</title>
    </head>
    <body>
        <itcast:prop times="5" date="<%=new Date()%>">kkkkkkkkkkkkl<br />
        </itcast:prop>
    </body>
</html>
```

```
    <tag>
        <name>prop</name>
        <tag-class>cn.itcast.web.simpletag.SimpleDemo5</tag-class>
        <body-content>scriptless</body-content>
        <attribute>
            <name>times</name>
            <required>yes</required>
        </attribute>
        <attribute>
            <name>date</name>
```

```
            <required>yes</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
```

```
package cn.itcast.web.simpletag;

import java.io.IOException;
import java.io.StringWriter;
import java.util.Date;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.SkipPageException;
import javax.servlet.jsp.tagext.JspFragment;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class SimpleDemo5 extends SimpleTagSupport {
    private int times;
    private Date date;

    public void setTimes(int times) {
        this.times = times;
    }
    public void setDate(Date date) {
        this.date = date;
    }
    @Override
    public void doTag() throws JspException, IOException {
        this.getJspContext().getOut().write(date.toString());
        for (int i=0;i<times;i++)
            this.getJspBody().invoke(null);
        //throw new SkipPageException();
    }
}
```

16.8 Tip：案例

使用标签控制页面逻辑案例：

16.9 开发防盗链标签

```
<%@ page language="java"    pageEncoding="UTF-8"%>
<%@taglib    uri ="/jsp2-tagexample" prefix="fix"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<fix:referer site="http://localhost" page="/index.jsp"></fix:referer>
<html>
  <head>
    <title>My JSP '1.jsp' starting page</title>
```

```
  </head>

  <body>
    This is my JSP page. <br>
  </body>
</html>
```

```
package cn.itcast.web.tag.eaxmple;

import java.io.IOException;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.SkipPageException;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class RefererTag extends SimpleTagSupport {
	private String page;
	private String site;
	public void setPage(String page) {
		this.page = page;
	}
	public void setSite(String site) {
		this.site = site;
	}
	@Override
	public void doTag() throws JspException, IOException {
		PageContext pageContext=(PageContext) this.getJspContext();
		HttpServletRequest request=(HttpServletRequest) pageContext.getRequest();
		HttpServletResponse response= (HttpServletResponse) pageContext.getResponse();
		String referer=request.getHeader("referer");

		if(referer==null || !referer.startsWith(site)){
			String cp=request.getContextPath();
			if(page.startsWith(cp))
			{	response.sendRedirect(page);
			}else if (page.startsWith("/")) {
				response.sendRedirect(cp+page);
			}else
			{
				response.sendRedirect(cp+"/"+page);
			}
			throw new SkipPageException();
		}
		else
```

```
                super.doTag();
        }

}
```

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

<taglib xmlns="http://java.sun.com/xml/ns/j2ee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-jsptaglibrary_2_0.xsd"
        version="2.0">
        <description>A tag library exercising SimpleTag handlers.</description>
        <tlib-version>1.0</tlib-version>
        <short-name>SimpleTagLibrary</short-name>
        <uri>/jsp2-tagexample</uri>

        <tag>
                <name>referer</name>
                <tag-class>cn.itcast.web.tag.eaxmple.RefererTag</tag-class>
                <body-content>empty</body-content>
                <attribute>
                        <name>site</name>
                        <required>yes</required>
                </attribute>
                <attribute>
                        <name>page</name>
                        <required>yes</required>
                        <rtexprvalue>true</rtexprvalue>
                </attribute>
        </tag>
</taglib>
```

16.10 开发<c:if>标签

```
public class IfTag extends SimpleTagSupport {
        private boolean test;

        public void setTest(boolean test) {
                this.test = test;
        }

        @Override
        public void doTag() throws JspException, IOException {
                // TODO Auto-generated method stub
                if(test)
```

```
                this.getJspBody().invoke(null);
            else
                super.doTag();

        }
}
```

```
    <body>
  <% session.setAttribute("user","mmmm") ;%>
    <fix:if test="${user==null }">
       未登陆. <br>
       </fix:if>
          <fix:if test="${user!=null }">
      welcome 用户已经登录. <br>
       </fix:if>

       ssss
    </body>
```

```
       <tag>
            <description>Outputs Hello, World</description>
            <name>if</name>
            <tag-class>cn.itcast.web.tag.example.IfTag</tag-class>
            <body-content>scriptless</body-content>
            <attribute>
                <name>test</name>
                <required>true</required>
                <rtexprvalue>true</rtexprvalue>
            </attribute>
       </tag>
```

16.11 开发<c:if><c:else>标签

```
<fix:choose>
       <fix:when test="${user==null }">
                                                    未登陆. <br>
       </fix:when>
       <fix:otherwith>
                    welcome 用户已经登录. <br>
       </fix:otherwith>
</fix:choose>
<fix:choose>
       <fix:when test="${2==2}">
            2==2<br>
       </fix:when>
       <fix:otherwith>
                2!=2<br>
       </fix:otherwith>
```

```
</fix:choose>

<fix:choose>
	<fix:when test="${1==1}">
		1==1 <br>
	</fix:when>
	<fix:otherwith>
			1<>1 <br>
	</fix:otherwith>
</fix:choose>
```

```
public class Choose extends SimpleTagSupport {
	private boolean isDo;
	public boolean isDo() {
		return isDo;
	}
	public void setDo(boolean isDo) {
		this.isDo = isDo;
	}
	@Override
	public void doTag() throws JspException, IOException {
		this.getJspBody().invoke(null);
	}
}
```

```
public class WhenTag extends SimpleTagSupport {

	private boolean test;

	public void setTest(boolean test) {
		this.test = test;
	}

	@Override
	public void doTag() throws JspException, IOException {
		Choose parent=(Choose) this.getParent();
		if(test && !parent.isDo())
		{
			this.getJspBody().invoke(null);
			parent.setDo(true);
		}
	}
}
```

```
public class OtherwithTag extends SimpleTagSupport {

	@Override
	public void doTag() throws JspException, IOException {
		Choose parent=(Choose) this.getParent();
```

```
            if(!parent.isDo())
            {
                this.getJspBody().invoke(null);
                parent.setDo(true);
            }
        }
}
```

16.12 开发迭代标签

```
<%@ page language="java"    pageEncoding="UTF-8" isELIgnored="false"%>
<%@page import="java.util.ArrayList"%>
<%@page import="java.util.List"%>
<%@taglib    uri ="/jsp2-tagexample" prefix="cc"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
   <head>
      <title>My JSP '1.jsp' starting page</title>
   </head>

<body>
<%
      List list=new ArrayList();
      list.add("aaa");
      list.add("bbb");
      list.add("ccc");
      list.add("ddd");
      request.setAttribute("list",list);
 %>
 <cc:foreach items="${list}" var="str">
 ${str }
 </cc:foreach>

</body>
</html>
```

```
<tag>
      <name>foreach</name>
      <tag-class>cn.itcast.web.tag.eaxmple.ForeachTag</tag-class>
      <body-content>scriptless</body-content>
      <attribute>
            <name>items</name>
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
      </attribute>
      <attribute>
            <name>var</name>
```

```
            <required>true</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
</tag>
```

```
package cn.itcast.web.tag.eaxmple;

import java.io.IOException;
import java.util.Iterator;
import java.util.List;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class ForeachTag extends SimpleTagSupport {

    private String var;
    private Object items;

    public void setVar(String var) {
        this.var = var;
    }

    public void setItems(Object items) {
        this.items = items;
    }

    @Override
    public void doTag() throws JspException, IOException {
        List list= (List) items;
        Iterator it=list.iterator();
        while(it.hasNext()){
            String value=(String) it.next();
            this.getJspContext().setAttribute(var, value);
            this.getJspBody().invoke(null);
        }
    }
}
```

```
<%@ page language="java"    pageEncoding="UTF-8" isELIgnored="false"%>
<%@page import="java.util.ArrayList"%>
<%@page import="java.util.List"%>
<%@page import="java.util.HashMap"%>
<%@page import="java.util.Map"%>
<%@taglib    uri ="/jsp2-tagexample" prefix="cc"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
```

```
<html>
  <head>
    <title>My JSP '1.jsp' starting page</title>
  </head>

<body>
_____list_____<br>
<%
      List list=new ArrayList();
      list.add("nnn");
      list.add("bbb");
      list.add("ccc");
      list.add("ddd");
      request.setAttribute("list",list);
 %>
 <cc:foreach2 items="${list}" var="str">
 ${str }<br/>
 </cc:foreach2>

  _____map_____<br>
 <%
      Map map=new HashMap();
      map.put("aa","111");
      map.put("bb","222");
      map.put("cc","333");
      request.setAttribute("map",map);
  %>
<cc:foreach2 items="${map}" var="entry">
  ${entry.key } :${entry.value }<br/>
</cc:foreach2>



_____Integer_____<br>
  <%
      Integer num[]={1,2,3,4};
      request.setAttribute("num",num);
    %>
<cc:foreach2 items="${num}" var="i">
${i }<br/>
</cc:foreach2>

_____String_____<br>
  <%
      String strs[]={"sss","mmm"};
      request.setAttribute("strs",strs);
    %>
<cc:foreach2 items="${strs}" var="str">
```

```
${str }<br/>
</cc:foreach2>
_____int_____<br>
   <%
      int num2[]={11,22,33,44};
      request.setAttribute("num2",num2);
    %>
<cc:foreach2 items="${num2}" var="i">
${i }
</cc:foreach2>
</body>
</html>
```

```
package cn.itcast.web.tag.eaxmple;

import java.io.IOException;
import java.lang.reflect.Array;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;
import java.util.Map;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class ForeachTag2 extends SimpleTagSupport {

	private String var;
	private Object items;
	private Collection collection;
	public void setVar(String var) {
		this.var = var;
	}

	public void setItems(Object items) {
		this.items = items;
		if(items==null)
			collection=null;
		if(items instanceof Collection)
		{
			collection=(Collection) items;
		}
		if(items instanceof Map){
			Map map=(Map)items;
```

```
                collection=map.entrySet();
            }
            if(items.getClass().isArray()){
                this.collection=new ArrayList();
                int len=Array.getLength(items);
                for(int i=0;i<len;i++)
                    this.collection.add(Array.get(items, i));
            }

/*          if(items instanceof Object[]){
                Object obj[]=(Object[]) items;
                collection=Arrays.asList(obj);
            }
            if(items instanceof int[]){
                int arr[]=(int[])items;
                this.collection=new ArrayList();
                for(int i:arr){
                    this.collection.add(i);
                }
            }*/
        }

        @Override
        public void doTag() throws JspException, IOException {
            if( collection==null) return;
            Iterator it=collection.iterator();
            while(it.hasNext()){
                Object value=it.next();
                this.getJspContext().setAttribute(var, value);
                this.getJspBody().invoke(null);
            }
        }
}
```

16.13 开发 html 转义标签

```
<fix:htmlFilter>
    <a href="a    发  a a">超链接的写法</a>
</fix:htmlFilter>
```

```
package cn.itcast.web.tag.example;

import java.io.IOException;
import java.io.StringWriter;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.JspFragment;
```

```
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class HtmlFilter extends SimpleTagSupport {

	@Override
	public void doTag() throws JspException, IOException {
		JspFragment jf=this.getJspBody();
		StringWriter sw=new StringWriter();
		jf.invoke(sw);
		String content=sw.getBuffer().toString();
		content=filter(content);
		this.getJspContext().getOut().write(content);
	}
	/*C:\\Tomcat 7.0\\webapps\\examples\\WEB-INF\\classes\\util\\HTMLFilter.java*/
	public static String filter(String message) {

		if (message == null)
			return (null);
		char content[] = new char[message.length()];
		message.getChars(0, message.length(), content, 0);
		StringBuilder result = new StringBuilder(content.length + 50);
		for (int i = 0; i < content.length; i++) {
			switch (content[i]) {
			case '<':
				result.append("&lt;");
				break;
			case '>':
				result.append("&gt;");
				break;
			case '&':
				result.append("&amp;");
				break;
			case '"':
				result.append("&quot;");
				break;
			default:
				result.append(content[i]);
			}
		}
		return (result.toString());
	}
}
```

16.14 打包标签库

1) 新建一 java 工程，把 src 目录下的东西的代码拷进来，再在工程目录下建立 META-INF 目录,把配置文件拷进来，点工程右键属性 export/Java/JAR file,next 在上面两个框中，左框中选中要导出的工程，去掉右框中 eclipse 的配置，在点 brower 按钮，输入导出的文件名，点 finish.

17 Tip：JSTL 标签库

核心标签
国际化标签
数据库标签
Xml 标签
Jstl 函数（el 函数）

17.1 Tip：<c:out>标签

<c:out> 标签用于输出一段文本内容到 pageContext 对象当前保存的“out”对象中。

属性名	是否支持 EL	属性类型	属 性 描 述
Value	true	Object	指定要输出的内容
escapeXml	true	Boolean	指定是否将>、<、&、'、" 等特殊字符进行 HTML 编码转换后再进行输出。默认值为 true
default	true	Object	指定如果 value 属性的值为 null 时所输出的默认值

```
<c:out value="aabbcc<br/>" escapeXml="false"></c:out>
 <c:out value="<a href=''>点点</a>" escapeXml="true"></c:out>
 <%
 request.setAttribute("data",null);
 %>
 <c:out default="bbbbbbbbbbbbb" value="${data }"></c:out>
```

17.2 Tip：<c:set>标签

<c:set>标签用于把某一个对象存在指定的域范围内，或者设置 Web 域中的 java.util.Map 类型的属性对象或 JavaBean 类型的属性对象的属性。

属性名	是否支持 EL	属性类型	属 性 描 述
value	true	Object	用于指定属性值
var	false	String	用于指定要设置的 Web 域属性的名称
scope	false	String	用于指定属性所在的 Web 域

target	true	Object	用于指定要设置属性的对象，这个对象必须是 JavaBean 对象或 java.util.Map 对象
property	true	string	用于指定当前要为对象设置的属性名称

```
<br/>_____c:set___可操作各个域 javabean Map 集合____
<c:set var="dd" value="ddvalue" scope="page"/>
${dd }<br/>

<% Map map=new HashMap();
      request.setAttribute("map",map);
 %>
<c:set property="dd" value="xxx" target="${map}" />
${map.dd }<br/>

<%
Person p=new Person();
request.setAttribute("p",p);
 %>
 <c:set property="name" value="namevalue" target="${p}"></c:set>
 <c:out value="${p.name}"></c:out>
```

```
package cn.itcast.domain;

public class Person {
      private String name;

      public String getName() {
            return name;
      }

      public void setName(String
name) {
            this.name = name;
      }

}
```

17.3 Tip：<c:remove>标签

<c:remove>标签用于删除各种 Web 域中的属性。

其语法格式如下：

```
<c:remove var="varName"
          [scope="{page|request|session|application}"] />
```

17.4 Tip：<c:catch>标签

<c:catch>标签用于捕获嵌套在标签体中的内容抛出的异常，其语法格式如下：<c:catch [var="varName"]>nested actions</c:catch>

var 属性用于标识<c:catch>标签捕获的异常对象，它将保存在 page 这个 Web 域中。

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page contentType="text/html;charset=gb2312" %>
<c:catch var="myex">
 <%
 int i=10/0;
  %>
 </c:catch>
异常：<c:out value="${myex}" /><br />
异常 myex.getMessage：<c:out value="${myex.message}" /><br />
异常 myex.getCause：<c:out value="${myex.cause}" /><br />
异常 myex.getStackTrace：<c:out value="${myex.stackTrace}" />
```

17.5 Tip：<c:if>标签

<c:if test=“”>标签可以构造简单的“if-then”结构的条件表达式

```
<c:if var="aaa" test="${user==null}" scope="page"></c:if>
```

属性名	是否支持 EL	属性类型	属 性 描 述
test	true	boolean	决定是否处理标签体中的内容的条件表达式
var	false	String	用于指定将 test 属性的执行结果保存到某个 Web 域中的某个属性的名称
scope	false	String	指定将 test 属性的执行结果保存到哪个 Web 域中

17.6 Tip：<c:choose>标签

<c:choose>标签用于指定多个条件选择的组合边界，它必须与<c:when>和<c:otherwise>标签一起使用。使用<c:choose>，<c:when>和<c:otherwise>三个标签，可以构造类似 “if-else if-else” 的复杂条件判断结构。

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ page contentType="text/html;charset=gb2312" %>
<c:set value="${param.count}" var="count“  /> pageContext(count,2)
<c:choose>
    <c:when test="${count == 0}">
        对不起，没有符合您要求的记录。
    </c:when>
    <c:otherwise>
        符合您要求的记录共有${count}条.
    </c:otherwise>
</c:choose>
```

17.7 Tip：<c:forEach>标签

<c:forEach>标签用于对一个集合对象中的元素进行循环迭代操作，或者按指定的次数重复迭代执行标签体中的内容。

属性名	是否支持 EL	属性类型	属 性 描 述
var	false	String	指定将当前迭代到的元素保存到 page 这个 Web 域中的属性名称
items	true	任何支持的类型	将要迭代的集合对象
varStatus	false	String	指定将代表当前迭代状态信息的对象保存到 page 这个 Web 域中的属性名称，是第几次接待
begin	true	int	如果指定 items 属性，就从集合中的第 begin 个元素开始进行迭代，begin 的索引值从 0 开始编号；如果没有指定 items 属性，就从 begin 指定的值开始迭代，直到 end 值时结束迭代
end	true	int	参看 begin 属性的描述

step	true	int	指定迭代的步长，即迭代因子的迭代增量

```
    <br/>_____________c:foreach_____
    <c:forEach var="i" begin="1" end="9" step="1" >
    ${i }
  </c:forEach>
  <br/>___c:foreach 实现表格隔行变色_____

  <% List list=new ArrayList();
     list.add("aaa");
     list.add("bbb");
     list.add("ccc");
     list.add("ddd");
     request.setAttribute("list",list);
 %>
 <style>
    .odd{background-color: blue;}
    .even{background-color: red;}
    tr:hover{background-color: green;}
    //如果不支持鼠标放上去变色，可从 tomact 默认网页上拷头<!DOCTYPE html>
 </style>
 <table>
 <c:forEach var="str" items="${list}" varStatus="status">
     <tr class="${status.count%2==0?'even':'odd' }"><td> ${status}    ::: ${str }</td></tr>
 </c:forEach>
 </table>
 <br/>_____________________________________
 <%
String atts[] = new String [5];
atts[0]="hello";
atts[1]="this";
atts[2]="is";
atts[3]="a";
atts[4]="pen";
request.setAttribute("atts", atts);
%>
<c:forEach items="${atts}" var="item" varStatus="s">
    <h2><c:out value="${item}"/> varStatus 的四种属性：</h2>
    index：${s.index}</br>
    count：${s.count}</br>
    first：${s.first}</br>
    last：${s.last}</br>
</c:forEach>
```

17.8 Tip：<c:param>标签

在 JSP 页面进行 URL 的相关操作时，经常要在 URL 地址后面附加一些参数。<c:param>标签可以嵌套在<c:import>、<c:url>或<c:redirect>标签内，为这些标签所使用的 URL 地址附加参数。<c:param>标签在为一个 URL 地址附加参数时，将自动对参数值进行 URL 编码，例如，如果传递的参数值为“中国”，则将其转换为“%d6%d0%b9%fa”后再附加到 URL 地址后面，这也就是使用<c:param>标签的最大好处。

http://localhost:808/servlet/MyServlet?name=“中国”
示例：<c:param name="name" value="value" />

17.9 Tip：<c:url>标签

<c:url>标签用于在 JSP 页面中构造一个 URL 地址，其主要目的是实现 URL 重写。URL 重写就是将会话标识号以参数形式附加在 URL 地址后面

属性名	是否支持 EL	属性类型	属 性 描 述
value	true	String	指定要构造的 URL
Context	true	String	当要使用相对路径导入同一个服务器下的其他 WEB 应用程序中的 URL 地址时，context 属性指定其他 WEB 应用程序的名称
var	false	String	指定将构造出的 URL 结果保存到 Web 域中的属性名称
scope	false	String	指定将构造出的 URL 结果保存到哪个 Web 域中

```
<c:url var="url" value="/index.jsp">
    <c:param name="name" value="中国"></c:param>
</c:url>
<a href="${url }">购买</a>
```

Tip：<c:redirect>标签
<c:redirect>标签用于将当前的访问请求转发或重定向到其他资源，它可以根据 url 属性所指定的地址，执行类似<jsp:forward>这个 JSP 标准标签的功能，将访问请求转发到其他资源；或执行 response.sendRedirect()方法的功能，将访问请求重定向到其他资源。

属性名	是否支持 EL	属性类型	属 性 描 述
url	true	String	指定要转发或重定向到的目标资源的 URL 地址
context	true	String	当要使用相对路径重定向到同一个服务器下的其他 WEB 应用程序中的资源时，context 属性指定其他 WEB 应用程序的名称

```
<% String data="aa,bb,cc,dd";
        request.setAttribute("data",data);
        %>
        <c:forTokens items="${data}" delims="\," var="c">
        ${c } :
        </c:forTokens>
```

18 EL 表达式

获得 web 开发常用对象

隐含对象名称	描　　述
pageContext	对应于 JSP 页面中的 pageContext 对象(注意:取的是 pageContext 对象。)
pageScope	代表 page 域中用于保存属性的 Map 对象
requestScope	代表 request 域中用于保存属性的 Map 对象
sessionScope	代表 session 域中用于保存属性的 Map 对象
applicationScope	代表 application 域中用于保存属性的 Map 对象
param	表示一个保存了所有请求参数的 Map 对象
paramValues	表示一个保存了所有请求参数的 Map 对象，它对于某个请求参数，返回的是一个 string[]
header	表示一个保存了所有 http 请求头字段的 Map 对象
headerValues	同上，返回 string[]数组。注意：如果头里面有“-”，例 Accept-Encoding，则要 headerValues[“Accept-Encoding”]
cookie	表示一个保存了所有 cookie 的 Map 对象
initParam	表示一个保存了所有 web 应用初始化参数的 map 对象

测试 headerValues 时，如果头里面有“-”，例 Accept-Encoding，则要 headerValues[“Accept-Encoding”]
测试 cookie 时，例${cookie.key}取的是 cookie 对象，如访问 cookie 的名称和值，须${cookie.key.name}或${cookie.key.value}
EL 表达式语法允许开发人员开发自定义函数，以调用 Java 类的方法。
示例：${prefix：method(params)}
在 EL 表达式中调用的只能是 Java 类的静态方法。
这个 Java 类的静态方法需要在 TLD 文件中描述，才可以被 EL 表达式调用。
EL 自定义函数用于扩展 EL 表达式的功能，可以让 EL 表达式完成普通 Java 程序代码所能完成的功能。

18.1 Tip：EL Function 开发步骤

一般来说， EL 自定义函数开发与应用包括以下三个步骤：
编写一个 Java 类的静态方法
编写标签库描述符（tld）文件，在 tld 文件中描述自定义函数。
在 JSP 页面中导入和使用自定义函数

示例：开发对 html 标签进行转义的 el function

18.2 开发 EL Function 注意事项

编写完标签库描述文件后，需要将它放置到<web 应用>\WEB-INF 目录中或 WEB-INF 目录下的除了 classes 和 lib 目录之外的任意子目录中。

TLD 文件中的<uri> 元素用指定该 TLD 文件的 URI，在 JSP 文件中需要通过这个 URI 来引入该标签库描述文件。
<function>元素用于描述一个 EL 自定义函数，其中：
<name>子元素用于指定 EL 自定义函数的名称。
<function-class>子元素用于指定完整的 Java 类名，
<function-signature>子元素用于指定 Java 类中的静态方法的签名，方法签名必须指明方法的返回值类型及各个参数的类型，各个参数之间用逗号分隔。

18.3 Tip：EL 注意事项

EL 表达式是 JSP 2.0 规范中的一门技术 。因此，若想正确解析 EL 表达式，需使用支持 Servlet2.4/JSP2.0 技术的 WEB 服务器。
注意：有些 Tomcat 服务器如不能使用 EL 表达式
（1）升级成 tomcat6
（2）在 JSP 中加入<%@ page isELIgnored="false" %>

18.4 Tip： EL 表达式保留关键字

And	eq	gt	true
Or	ne	le	false
No	lt	ge	null
instanceof	empty	div	mod

${sessionScope.user[data]} //data 是一个变量 .无法做到动态取值

18.5 Tip：JSTL 中的常用 EL 函数

由于在 JSP 页面中显示数据时，经常需要对显示的字符串进行处理，SUN 公司针对于一些常见处理定义了一套 EL 函数库供开发者使用。
这些 EL 函数在 JSTL 开发包中进行描述，因此在 JSP 页面中使用 SUN 公司的 EL 函数库，需要导入 JSTL 开发包，并在页面中导入 EL 函数库，如下所示：
在页面中使用 JSTL 定义的 EL 函数：
<%@taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%>
fn:toLowerCase 函数将一个字符串中包含的所有字符转换为小写形式，并返回转换后的字符串，它接收一个字符串类型的参数，例如
fn:toLowerCase("Www.IT315.org") 的返回值为字符串“www.it315.org”
fn:toLowerCase("")的返回值为空字符串
fn:toUpperCase 函数将一个字符串中包含的所有字符转换为大写形式，并返回转换后的字符串，它接收一个字符串类型的参数。例如：
fn:toUpperCase("Www.IT315.org") 的返回值为字符串“WWW.IT315.ORG”
fn:toUpperCase("")的返回值为空字符串
fn:trim 函数删除一个字符串的首尾的空格，并返回删除空格后的结果字符串，它接收一个字符串类型的参数。需要注意的是，fn:trim 函数不能删除字符串中间位置的空格。
例如，fn:trim(" www.it315.org ") 的返回值为字符串“www.it 315.org”。
fn:length 函数返回一个集合或数组大小，或返回一个字符串中包含的字符的个数，返回值为 int 类型。fn:length 函数接收一个参数，这个参数可以是<c:forEach>标签的 items 属性支持的任何类型，包括任意类型的数组、java.util.Collection、java.util.Iterator、java.util.Enumeration、java.util.Map 等类的实例对象和字符串。

如果 fn:length 函数的参数为 null 或者是元素个数为 0 的集合或数组对象，则函数返回 0；如果参数是空字符串，则函数返回 0。
fn:split 函数以指定字符串作为分隔符，将一个字符串分割成字符串数组并返回这个字符串数组。

fn:split 函数接收两个字符串类型的参数，第一个参数表示要分割的字符串，第二个参数表示作为分隔符的字符串。

例如，fn:split("www.it315.org", ".")[1]的返回值为字符串“it315”。
fn:join 函数以一个字符串作为分隔符，将一个字符串数组中的所有元素合并为一个字符串并返回合并后的结果字符串。fn:join 函数接收两个参数，第一个参数是要操作的字符串数组，第二个参数是作为分隔符的字符串。
如果 fn:join 函数的第二个参数是空字符串，则 fn:join 函数的返回值直接将元素连接起来。例如：
假设 stringArray 是保存在 Web 域中的一个属性，它表示一个值为{"www","it315","org"}的字符串数组，则 fn:join(stringArray, “.")返回字符串“www.it315.org”
fn:join(fn:split("www,it315,org", ","), ".") 的返回值为字符串 www.it315.org
fn:indexOf 函数返回指定字符串在一个字符串中第一次出现的索引值，返回值为 int 类型。fn:indexOf 函数接收两个字符串类型的参数，如果第一个参数字符串中包含第二个参数字符串，那么，不管第二个参数字符串在第一个参数字符串中出现几次，fn:indexOf 函数总是返回第一次出现的索引值；如果第一个参数中不包含第二个参数，则 fn:indexOf 函数返回-1。如果第二个参数为空字符串，则 fn:indexOf 函数总是返回 0。例如：
fn:indexOf("www.it315.org","t3") 的返回值为 5
fn:contains 函数检测一个字符串中是否包含指定的字符串，返回值为布尔类型。fn:contains 函数在比较两个字符串是否相等时是大小写敏感的。
fn:contains 函数接收两个字符串类型的参数，如果第一个参数字符串中包含第二个参数字符串，则 fn:contains 函数返回 true，否则返回 false。如果第二个参数的值为空字符串，则 fn:contains 函数总是返回 true。实际上，fn:contains(string, substring)等价于 fn:indexOf(string, substring) != -1。

忽略大小的 EL 函数：fn:containsIgnoreCase
fn:startsWith 函数用于检测一个字符串是否是以指定字符串开始的，返回值为布尔类型。

fn:startsWith 函数接收两个字符串类型的参数，如果第一个参数字符串以第二个参数字符串开始，则函数返回 true，否则函数返回 false。如果第二个参数为空字符串，则 fn:startsWith 函数总是返回 true。例如：

fn:startsWith("www.it315.org","it315")的返回值为 false

与之对应的 EL 函数：fn:endsWith
fn:replace 函数将一个字符串中包含的指定子字符串替换为其它的指定字符串，并返回替换后的结果字符串。fn:replace 方法接收三个字符串类型的参数，第一个参数表示要操作的源字符串，第二个参数表示源字符串中要被替换的子字符串，第三个参数表示要被替换成的字符串。例如：

fn:replace("www it315 org", " ", ".")的返回值为字符串 www.it315.org
fn:substring 函数用于截取一个字符串的子字符串并返回截取到的子字符串。fn:substring 函数接收三个参数，第一个参数是用于指定要操作的源字符串，第二个参数是用于指定截取子字符串开始的索引值，第三个参数是用于指定截取子字符串结束的索引值，第二个参数和第三个参数都是 int 类型，其值都从 0 开始。例如：

fn:substring("www.it315.org", 4, 9) 的返回值为字符串“it315”
fn:substringAfter 函数用于截取并返回一个字符串中的指定子字符串第一次出现之后的子字符串。fn:substringAfter 函数接收两个字符串类型的参数，第一个参数表示要操作的源字符串，第二个参数表示指定的子字符串，例如：

fn:substringAfter(“www.it315.org”, “.”)的返回值为字符串“it315.org”。

与之对应的 EL 函数为：fn:substringBefore

19 Tip：Filter 简介

Filter 也称之为过滤器，它是 Servlet 技术中最激动人心的技术，WEB 开发人员通过 Filter 技术，对 web 服务器管理的所有 web 资源：例如 Jsp, Servlet, 静态图片文件或静态 html 文件等进行拦截，从而实现一些特殊的功能。例如实现 URL 级别的权限访问控制、过滤敏感词汇、压缩响应信息等一些高级功能。
Servlet API 中提供了一个 Filter 接口，开发 web 应用时，如果编写的 Java 类实现了这个接口，则把这个 java 类称之为过滤器 Filter。通过 Filter 技术，开发人员可以实现用户在访问某个目标资源之前，对访问的请求和响应进行拦截，如下所示：

19.1 Tip：Filter 是如何实现拦截的？

Filter 接口中有一个 doFilter 方法，当开发人员编写好 Filter，并配置对哪个 web 资源进行拦截后，WEB 服务器每次在调用 web 资源的 service 方法之前，都会先调用一下 filter 的 doFilter 方法，因此，在该方法内编写代码可达到如下目的：
调用目标资源之前，让一段代码执行
是否调用目标资源（即是否让用户访问 web 资源）。
web 服务器在调用 doFilter 方法时，会传递一个 filterChain 对象进来，filterChain 对象是 filter 接口中最重要的一个对象，它也提供了一个 doFilter 方法，开发人员可以根据需求决定是否调用此方法，调用该方法，则 web 服务器就会调用 web 资源的 service 方法，即 web 资源就会被访问，否则 web 资源不会被访问。
调用目标资源之后，让一段代码执行
实验： Filter 开发，见下页 PPT 中的开发流程

19.2 Tip：Filter 开发入门

Filter 开发分为二个步骤：
编写 java 类实现 Filter 接口，并实现其 doFilter 方法。
在 web.xml 文件中使用<filter>和<filter-mapping>元素对编写的 filter 类进行注册，并设置它所能拦截的资源。（动手实验）
Filter 链
在一个 web 应用中，可以开发编写多个 Filter，这些 Filter 组合起来称之为一个 Filter 链。
web 服务器根据 Filter 在 web.xml 文件中的注册顺序，决定先调用哪个 Filter，当第一个 Filter 的 doFilter 方法被调用时，web 服务器会创建一个代表 Filter 链的 FilterChain 对象传递给该方法。在 doFilter 方法中，开发人员如果调用了 FilterChain 对象的 doFilter 方法，则 web 服务器会检查 FilterChain 对象中是否还有 filter，如果有，则调用第 2 个 filter，如果没有，则调用目标资源。
Filter 链实验（查看 filterChain API 文档）

19.3 Tip： Filter 的生命周期

init(FilterConfig filterConfig)throws ServletException：
和我们编写的 Servlet 程序一样，Filter 的创建和销毁由 WEB 服务器负责。 web 应用程序启动时，web 服务器将创建 Filter 的实例对象，并调用其 init 方法，完成对象的初始化功能，从而为后续的用户请求作好拦截的准备工作（注：filter 对象只会创建一次，init 方法也只会执行一次。示例 ）

开发人员通过 init 方法的参数，可获得代表当前 filter 配置信息的 FilterConfig 对象。(filterConfig 对象见下页 PPT)
destroy()：
在 Web 容器卸载 Filter 对象之前被调用。该方法在 Filter 的生命周期中仅执行一次。在这个方法中，可以释放过滤器使用的资源。

19.4 Tip： FilterConfig 接口

用户在配置 filter 时，可以使用<init-param>为 filter 配置一些初始化参数，当 web 容器实例化 Filter 对象，调用其 init 方法时，会把封装了 filter 初始化参数的 filterConfig 对象传递进来。因此开发人员在编写 filter 时，通过 filterConfig 对象的方法，就可获得：
String getFilterName()：得到 filter 的名称。
String getInitParameter(String name)： 返回在部署描述中指定名称的初始化参数的值。如果不存在返回 null.
Enumeration getInitParameterNames()：返回过滤器的所有初始化参数的名字的枚举集合。
public ServletContext getServletContext()：返回 Servlet 上下文对象的引用。
实验：得到 filter 配置信息

19.5 Tip：Filter 常见应用(1)

统一全站字符编码的过滤器
通过配置参数 encoding 指明使用何种字符编码,以处理 Html Form 请求参数的中文问题
禁止浏览器缓存所有动态页面的过滤器：
有 3 个 HTTP 响应头字段都可以禁止浏览器缓存当前页面，它们在 Servlet 中的示例代码如下：

```
response.setDateHeader("Expires",-1);
response.setHeader("Cache-Control","no-cache");
response.setHeader("Pragma","no-cache");
```

并不是所有的浏览器都能完全支持上面的三个响应头，因此最好是同时使用上面的三个响应头。
Expires 数据头：值为 GMT 时间值，为-1 指浏览器不要缓存页面
Cache-Control 响应头有两个常用值：
no-cache 指浏览器不要缓存当前页面。
max-age:xxx 指浏览器缓存页面 xxx 秒。
控制浏览器缓存页面中的静态资源的过滤器：
场景：有些动态页面中引用了一些图片或 css 文件以修饰页面效果，这些图片和 css 文件经常是不变化的，所以为减轻服务器的压力，可以使用 filter 控制浏览器缓存这些文件，以提升服务器的性能。
使用 Filter 实现 URL 级别的权限认证
情景：在实际开发中我们经常把一些执行敏感操作的 servlet 映射到一些特殊目录中，并用 filter 把这些特殊目录保护起来，限制只能拥有相应访问权限的用户才能访问这些目录下的资源。从而在我们系统中实现一种 URL 级别的权限功能。
要求：为使 Filter 具有通用性，Filter 保护的资源和相应的访问权限通过 filter 参数的形式予以配置。
实现用户自动登陆的过滤器
在用户登陆成功后，发送一个名称为 user 的 cookie 给客户端，cookie 的值为用户名和 md5 加密后的密码。
编写一个 AutoLoginFilter，这个 filter 检查用户是否带有名称为 user 的 cookie 来，如果有，则调用 dao 查询 cookie 的用户名和密码是否和数据库匹配，匹配则向 session 中存入 user 对象（即用户登陆标记），以实现程序完成自动登陆。

19.6 Tip：Filter 的部署—注册 Filter

```
<filter>
        <filter-name>testFitler</filter-name>
        <filter-class>org.test.TestFiter</filter-class>
```

```
        <init-param>
        <param-name>word_file</param-name>
        <param-value>/WEB-INF/word.txt</param-value>
        </init-param>
</filter>
```

<filter-name>用于为过滤器指定一个名字，该元素的内容不能为空。
<filter-class>元素用于指定过滤器的完整的限定类名。
<init-param>元素用于为过滤器指定初始化参数，它的子元素<param-name>指定参数的名字，<param-value>指定参数的值。在过滤器中，可以使用 FilterConfig 接口对象来访问初始化参数。

19.7 Tip：Filter 的部署—映射 Filter

<filter-mapping>元素用于设置一个 Filter 所负责拦截的资源。一个 Filter 拦截的资源可通过两种方式来指定：Servlet 名称和资源访问的请求路径
<filter-name>子元素用于设置 filter 的注册名称。该值必须是在<filter>元素中声明过的过滤器的名字
<url-pattern>设置 filter 所拦截的请求路径(过滤器关联的 URL 样式)
<servlet-name>指定过滤器所拦截的 Servlet 名称。
<dispatcher>指定过滤器所拦截的资源被 Servlet 容器调用的方式，可以是 REQUEST,INCLUDE,FORWARD 和 ERROR 之一，默认 REQUEST。用户可以设置多个<dispatcher> 子元素用来指定 Filter 对资源的多种调用方式进行拦截。
<dispatcher> 子元素可以设置的值及其意义：
REQUEST：当用户直接访问页面时，Web 容器将会调用过滤器。如果目标资源是通过 RequestDispatcher 的 include()或 forward()方法访问时，那么该过滤器就不会被调用。
INCLUDE：如果目标资源是通过 RequestDispatcher 的 include()方法访问时，那么该过滤器将被调用。除此之外，该过滤器不会被调用。
FORWARD：如果目标资源是通过 RequestDispatcher 的 forward()方法访问时，那么该过滤器将被调用，除此之外，该过滤器不会被调用。
ERROR：如果目标资源是通过声明式异常处理机制调用时，那么该过滤器将被调用。除此之外，过滤器不会被调用。

19.8 Tip：Filter 的部署—映射 Filter 示例

```
<filter-mapping>
       <filter-name>testFilter</filter-name>
      <url-pattern>/test.jsp</url-pattern>
</filter-mapping>
```

```
<filter-mapping>
      <filter-name>testFilter</filter-name>
    <url-pattern>/index.jsp</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

19.9 Tip：Filter 高级开发

由于开发人员在 filter 中可以得到代表用户请求和响应的 request、response 对象，因此在编程中可以使用 Decorator(装饰器)模式对 request、response 对象进行包装，再把包装对象传给目标资源，从而实现一些特殊需求。

19.10 Tip： Decorator 设计模式

某个对象的方法不适应业务需求时，通常有 2 种方式可以对方法进行增强：
编写子类，覆盖需增强的方法
使用 Decorator 设计模式对方法进行增强
疑问：在实际应用中遇到需增强对象的方法时，到底选用哪种方式呢？
没有具体的定式，不过有一种情况下，必须使用 Decorator 设计模式：即被增强的对象，开发人员只能得到它的对象，无法得到它的 class 文件。
比如 request、response 对象，开发人员之所以在 servlet 中能通过 sun 公司定义的 HttpServletRequest\response 接口去操作这些对象，是因为 Tomcat 服务器厂商编写了 request、response 接口的实现类。web 服务器在调用 servlet 时，会用这些接口的实现类创建出对象，然后传递给 servlet 程序。
此种情况下，由于开发人员根本不知道服务器厂商编写的 request、response 接口的实现类是哪个？在程序中只能拿到服务器厂商提供的对象，因此就只能采用 Decorator 设计模式对这些对象进行增强。
Decorator 设计模式的实现
1.首先看需要被增强对象继承了什么接口或父类，编写一个类也去继承这些接口或父类。
2.在类中定义一个变量，变量类型即需增强对象的类型。
3.在类中定义一个构造函数，接收需增强的对象。
4.覆盖需增强的方法，编写增强的代码。
举例：使用 Decorator 设计模式为 BufferedReader 类的 readLine 方法添加行号的功能。

19.11 Tip：request 对象的增强

Servlet API 中提供了一个 request 对象的 Decorator 设计模式的默认实现类 HttpServletRequestWrapper ，（HttpServletRequestWrapper 类实现了 request 接口中的所有方法，但这些方法的内部实现都是仅仅调用了一下所包装的的 request 对象的对应方法）以避免用户在对 request 对象进行增强时需要实现 request 接口中的所有方法。
使用 Decorator 模式包装 request 对象，完全解决 get、post 请求方式下的乱码问题。

使用 Decorator 模式包装 request 对象，实现 html 标签转义功能（Tomcat 服务器中提供了转义 html 标签的工具类）。

19.12 Tip： response 对象的增强

Servlet API 中提供了 response 对象的 Decorator 设计模式的默认实现类 HttpServletResponseWrapper ，（HttpServletResponseWrapper 类实现了 response 接口中的所有方法，但这些方法的内部实现都是仅仅调用了一下所包装的的 response 对象的对应方法）以避免用户在对 response 对象进行增强时需要实现 response 接口中的所有方法。

19.13 Tip：response 增强案例—压缩响应

应用 HttpServletResponseWrapper 对象，压缩响应正文内容。思路:
通过 filter 向目标页面传递一个自定义的 response 对象。
在自定义的 response 对象中，重写 getOutputStream 方法和 getWriter 方法，使目标资源调用此方法输出页面内容时，获得的是我们自定义的 ServletOutputStream 对象。
在我们自定义的 ServletOuputStream 对象中，重写 write 方法，使写出的数据写出到一个 buffer 中。
当页面完成输出后，在 filter 中就可得到页面写出的数据，从而我们可以调用 GzipOuputStream 对数据进行压缩后再写出给浏览器，以此完成响应正文件压缩功能。

19.14 Tip：实用案例－缓存数据到内存

对于页面中很少更新的数据，例如商品分类，为避免每次都要从数据库查询分类数据，因此可把分类数据缓存在内存或文件中，以此来减轻数据库压力，提高系统响应速度。

20 Tip：动态代理

在 java 里，每个对象都有一个类与之对应。
现在要生成某一个对象的代理对象，这个代理对象也要通过一个类来生成，所以首先要编写用于生成代理对象的类。
如何编写生成代理对象的类，两个要素：
代理谁
如何生成代理对象
代理谁？
设计一个类变量，以及一个构造函数，记住代理类 代理哪个对象。
如何生成代理对象？
设计一个方法生成代理对象（在方法内编写代码生成代理对象是此处编程的难点）
Java 提供了一个 Proxy 类，调用它的 newInstance 方法可以生成某个对象的代理对象，使用该方法生成代理对象时，需要三个参数：
1.生成代理对象使用哪个类装载器
2.生成哪个对象的代理对象，通过接口指定
3.生成的代理对象的方法里干什么事，由开发人员编写 handler 接口的实现来指定。
初学者必须理解，或不理解必须记住的 2 件事情：
Proxy 类负责创建代理对象时，如果指定了 handler（处理器），那么不管用户调用代理对象的什么方法，该方法都是调用处理器的 invoke 方法。
由于 invoke 方法被调用需要三个参数：代理对象、方法、方法的参数，因此不管代理对象哪个方法调用处理器的 invoke 方法，都必须把自己所在的对象、自己（调用 invoke 方法的方法）、方法的参数传递进来。

20.1 Tip：动态代理应用

在动态代理技术里，由于不管用户调用代理对象的什么方法，都是调用开发人员编写的处理器的 invoke 方法（这相当于 invoke 方法拦截到了代理对象的方法调用）。
并且，开发人员通过 invoke 方法的参数，才可以在拦截的同时，知道用户调用的是什么方法，因此利用这两个特性，就可以实现一些特殊需求，例如：拦截用户的访问请求，以检查用户是否有访问权限、动态为某个对象添加额外的功能。

20.2 Filter 039 张龙

1. 过滤器 单例模式
   1） 过滤器本身并不生成请求和响应对象，它只提供过滤作用。
   2） Servlet filter 能够在 servlet 被调用之前检查 request 对象，修改 request 和 ader 和 request 内容；
   3） 在 servlet 被调用之后检查 response 对象，修改 response header 和 response 内容。Servlet filter 负责过滤的 web 组件可以是 servlet jsp 或 html 文件。

有一个 filter 启动不了，整个 web 应用就启动不了。

```
package compressionFilters;

import java.io.IOException;
import java.io.OutputStream;
import java.util.zip.GZIPOutputStream;

import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServletResponse;


/**
 * Implementation of <b>ServletOutputStream</b> that works with
 * the CompressionServletResponseWrapper implementation.
* @version $Id: CompressionResponseStream.java 987920 2010-08-22 15:34:34Z markt $
 */

public class CompressionResponseStream
    extends ServletOutputStream {

    // ----------------------------------------------------------- Constructors

    /**
     * Construct a servlet output stream associated with the specified Response.
     *
     * @param response The associated response
     */
    public CompressionResponseStream(HttpServletResponse response) throws IOException{

        super();
        closed = false;
        this.response = response;
        this.output = response.getOutputStream();

    }

    // ----------------------------------------------------- Instance Variables

    /**
     * The threshold number which decides to compress or not.
     * Users can configure in web.xml to set it to fit their needs.
     */
    protected int compressionThreshold = 0;

    /**
     * Debug level
     */
    private int debug = 0;
```

```
/**
 * The buffer through which all of our output bytes are passed.
 */
protected byte[] buffer = null;

/**
 * The number of data bytes currently in the buffer.
 */
protected int bufferCount = 0;

/**
 * The underlying gzip output stream to which we should write data.
 */
protected OutputStream gzipstream = null;

/**
 * Has this stream been closed?
 */
protected boolean closed = false;

/**
 * The content length past which we will not write, or -1 if there is
 * no defined content length.
 */
protected int length = -1;

/**
 * The response with which this servlet output stream is associated.
 */
protected HttpServletResponse response = null;

/**
 * The underlying servket output stream to which we should write data.
 */
protected ServletOutputStream output = null;


// ------------------------------------------------------- Public Methods

/**
 * Set debug level
 */
public void setDebugLevel(int debug) {
      this.debug = debug;
}
```

```
    /**
     * Set the compressionThreshold number and create buffer for this size
     */
    protected void setBuffer(int threshold) {
        compressionThreshold = threshold;
        buffer = new byte[compressionThreshold];
        if (debug > 1) {
            System.out.println("buffer is set to "+compressionThreshold);
        }
    }

    /**
     * Close this output stream, causing any buffered data to be flushed and
     * any further output data to throw an IOException.
     */
    @Override
    public void close() throws IOException {

        if (debug > 1) {
            System.out.println("close() @ CompressionResponseStream");
        }
        if (closed)
            throw new IOException("This output stream has already been closed");

        if (gzipstream != null) {
            flushToGZip();
            gzipstream.close();
            gzipstream = null;
        } else {
            if (bufferCount > 0) {
                if (debug > 2) {
                    System.out.print("output.write(");
                    System.out.write(buffer, 0, bufferCount);
                    System.out.println(")");
                }
                output.write(buffer, 0, bufferCount);
                bufferCount = 0;
            }
        }

        output.close();
        closed = true;

    }

    /**
     * Flush any buffered data for this output stream, which also causes the
```

```
     * response to be committed.
     */
    @Override
    public void flush() throws IOException {

        if (debug > 1) {
            System.out.println("flush() @ CompressionResponseStream");
        }
        if (closed) {
            throw new IOException("Cannot flush a closed output stream");
        }

        if (gzipstream != null) {
            gzipstream.flush();
        }

    }

    public void flushToGZip() throws IOException {

        if (debug > 1) {
            System.out.println("flushToGZip() @ CompressionResponseStream");
        }
        if (bufferCount > 0) {
            if (debug > 1) {
                System.out.println("flushing out to GZipStream, bufferCount = " + bufferCount);
            }
            writeToGZip(buffer, 0, bufferCount);
            bufferCount = 0;
        }

    }

    /**
     * Write the specified byte to our output stream.
     *
     * @param b The byte to be written
     *
     * @exception IOException if an input/output error occurs
     */
    @Override
    public void write(int b) throws IOException {

        if (debug > 1) {
            System.out.println("write "+b+" in CompressionResponseStream ");
        }
        if (closed)
```

```
            throw new IOException("Cannot write to a closed output stream");

        if (bufferCount >= buffer.length) {
            flushToGZip();
        }

        buffer[bufferCount++] = (byte) b;

    }


    /**
     * Write <code>b.length</code> bytes from the specified byte array
     * to our output stream.
     *
     * @param b The byte array to be written
     *
     * @exception IOException if an input/output error occurs
     */
    @Override
    public void write(byte b[]) throws IOException {

        write(b, 0, b.length);

    }


    /**
     * Write <code>len</code> bytes from the specified byte array, starting
     * at the specified offset, to our output stream.
     *
     * @param b The byte array containing the bytes to be written
     * @param off Zero-relative starting offset of the bytes to be written
     * @param len The number of bytes to be written
     *
     * @exception IOException if an input/output error occurs
     */
    @Override
    public void write(byte b[], int off, int len) throws IOException {

        if (debug > 1) {
            System.out.println("write, bufferCount = " + bufferCount + " len = " + len + " off = " + off);
        }
        if (debug > 2) {
            System.out.print("write(");
            System.out.write(b, off, len);
            System.out.println(")");
```

```
        }

        if (closed)
            throw new IOException("Cannot write to a closed output stream");

        if (len == 0)
            return;

        // Can we write into buffer ?
        if (len <= (buffer.length - bufferCount)) {
            System.arraycopy(b, off, buffer, bufferCount, len);
            bufferCount += len;
            return;
        }

        // There is not enough space in buffer. Flush it ...
        flushToGZip();

        // ... and try again. Note, that bufferCount = 0 here !
        if (len <= (buffer.length - bufferCount)) {
            System.arraycopy(b, off, buffer, bufferCount, len);
            bufferCount += len;
            return;
        }

        // write direct to gzip
        writeToGZip(b, off, len);
    }

    public void writeToGZip(byte b[], int off, int len) throws IOException {

        if (debug > 1) {
            System.out.println("writeToGZip, len = " + len);
        }
        if (debug > 2) {
            System.out.print("writeToGZip(");
            System.out.write(b, off, len);
            System.out.println(")");
        }
        if (gzipstream == null) {
            if (debug > 1) {
                System.out.println("new GZIPOutputStream");
            }
            if (response.isCommitted()) {
                if (debug > 1)
                    System.out.print("Response already committed. Using original output stream");
                gzipstream = output;
```

```
                } else {
                    response.addHeader("Content-Encoding", "gzip");
                    String vary = response.getHeader("Vary");
                    if (vary == null) {
                        // Add a new Vary header
                        response.setHeader("Vary", "Accept-Encoding");
                    } else if (vary.equals("*")) {
                        // No action required
                    } else {
                        // Merge into current header
                        response.setHeader("Vary", vary + ",Accept-Encoding");
                    }
                    gzipstream = new GZIPOutputStream(output);
                }
            }
            gzipstream.write(b, off, len);

        }

        // ------------------------------------------------------- Package Methods

        /**
         * Has this response stream been closed?
         */
        public boolean closed() {

            return (this.closed);

        }

}
```

```
package compressionFilters;

import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;

import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletResponseWrapper;

/**
 * Implementation of <b>HttpServletResponseWrapper</b> that works with
 * the CompressionServletResponseStream implementation..
* @version $Id: CompressionServletResponseWrapper.java 987920 2010-08-22 15:34:34Z markt $
 */
```

```
public class CompressionServletResponseWrapper extends HttpServletResponseWrapper {

    // ----------------------------------------------------- Constructor

    /**
     * Calls the parent constructor which creates a ServletResponse adaptor
     * wrapping the given response object.
     */

    public CompressionServletResponseWrapper(HttpServletResponse response) {
        super(response);
        origResponse = response;
        if (debug > 1) {
            System.out.println("CompressionServletResponseWrapper constructor gets called");
        }
    }

    // ----------------------------------------------------- Instance Variables
    /**
     * Original response
     */

    protected HttpServletResponse origResponse = null;

    /**
     * Descriptive information about this Response implementation.
     */

    protected static final String info = "CompressionServletResponseWrapper";

    /**
     * The ServletOutputStream that has been returned by
     * <code>getOutputStream()</code>, if any.
     */

    protected ServletOutputStream stream = null;


    /**
     * The PrintWriter that has been returned by
     * <code>getWriter()</code>, if any.
     */

    protected PrintWriter writer = null;

    /**
```

```
     * The threshold number to compress
     */
    protected int threshold = 0;

    /**
     * Debug level
     */
    private int debug = 0;

    /**
     * Content type
     */
    protected String contentType = null;

    // ----------------------------------------------------------- Public Methods

    /**
     * Set content type
     */
    @Override
    public void setContentType(String contentType) {
        if (debug > 1) {
            System.out.println("setContentType to "+contentType);
        }
        this.contentType = contentType;
        origResponse.setContentType(contentType);
    }


    /**
     * Set threshold number
     */
    public void setCompressionThreshold(int threshold) {
        if (debug > 1) {
            System.out.println("setCompressionThreshold to " + threshold);
        }
        this.threshold = threshold;
    }


    /**
     * Set debug level
     */
    public void setDebugLevel(int debug) {
        this.debug = debug;
    }
```

```
    /**
     * Create and return a ServletOutputStream to write the content
     * associated with this Response.
     *
     * @exception IOException if an input/output error occurs
     */
    public ServletOutputStream createOutputStream() throws IOException {
        if (debug > 1) {
            System.out.println("createOutputStream gets called");
        }

        CompressionResponseStream compressedStream =
            new CompressionResponseStream(origResponse);
        compressedStream.setDebugLevel(debug);
        compressedStream.setBuffer(threshold);

        return compressedStream;

    }

    /**
     * Finish a response.
     */
    public void finishResponse() {
        try {
            if (writer != null) {
                writer.close();
            } else {
                if (stream != null)
                    stream.close();
            }
        } catch (IOException e) {
            // Ignore
        }
    }

    // ---------------------------------------------- ServletResponse Methods

    /**
     * Flush the buffer and commit this response.
     *
     * @exception IOException if an input/output error occurs
     */
    @Override
    public void flushBuffer() throws IOException {
        if (debug > 1) {
            System.out.println("flush buffer @ CompressionServletResponseWrapper");
```

```
            }
            ((CompressionResponseStream)stream).flush();

        }

        /**
         * Return the servlet output stream associated with this Response.
         *
         * @exception IllegalStateException if <code>getWriter</code> has
         *  already been called for this response
         * @exception IOException if an input/output error occurs
         */
        @Override
        public ServletOutputStream getOutputStream() throws IOException {

            if (writer != null)
                throw new IllegalStateException("getWriter() has already been called for this response");

            if (stream == null)
                stream = createOutputStream();
            if (debug > 1) {
                System.out.println("stream is set to "+stream+" in getOutputStream");
            }

            return (stream);

        }

        /**
         * Return the writer associated with this Response.
         *
         * @exception IllegalStateException if <code>getOutputStream</code> has
         *  already been called for this response
         * @exception IOException if an input/output error occurs
         */
        @Override
        public PrintWriter getWriter() throws IOException {

            if (writer != null)
                return (writer);

            if (stream != null)
                throw new IllegalStateException("getOutputStream() has already been called for this
response");

            stream = createOutputStream();
            if (debug > 1) {
```

```
                System.out.println("stream is set to "+stream+" in getWriter");
            }
            //String charset = getCharsetFromContentType(contentType);
            String charEnc = origResponse.getCharacterEncoding();
            if (debug > 1) {
                System.out.println("character encoding is " + charEnc);
            }
            // HttpServletResponse.getCharacterEncoding() shouldn't return null
            // according the spec, so feel free to remove that "if"
            if (charEnc != null) {
                writer = new PrintWriter(new OutputStreamWriter(stream, charEnc));
            } else {
                writer = new PrintWriter(stream);
            }

            return (writer);

        }

        @Override
        public void setContentLength(int length) {
            // Don't, as compression will change it
        }

}
```

```
package compressionFilters;

import java.io.IOException;
import java.util.Enumeration;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


/**
 * Implementation of <code>javax.servlet.Filter</code> used to compress
 * the ServletResponse if it is bigger than a threshold.
* @version $Id: CompressionFilter.java 987920 2010-08-22 15:34:34Z markt $
 */
```

```
public class CompressionFilter implements Filter{

    /**
     * The filter configuration object we are associated with.    If this value
     * is null, this filter instance is not currently configured.
     */
    private FilterConfig config = null;

    /**
     * Minimal reasonable threshold
     */
    private int minThreshold = 128;


    /**
     * The threshold number to compress
     */
    protected int compressionThreshold;

    /**
     * Debug level for this filter
     */
    private int debug = 0;

    /**
     * Place this filter into service.          *
     * @param filterConfig The filter configuration object
     */

    @Override
    public void init(FilterConfig filterConfig) {

        config = filterConfig;
        if (filterConfig != null) {
            String value = filterConfig.getInitParameter("debug");
            if (value!=null) {
                debug = Integer.parseInt(value);
            } else {
                debug = 0;
            }
            String str = filterConfig.getInitParameter("compressionThreshold");
            if (str!=null) {
                compressionThreshold = Integer.parseInt(str);
                if (compressionThreshold != 0 && compressionThreshold < minThreshold) {
                    if (debug > 0) {
                        System.out.println("compressionThreshold should be either 0 - no
compression or >= " + minThreshold);
```

```
                        System.out.println("compressionThreshold set to " + minThreshold);
                    }
                    compressionThreshold = minThreshold;
                }
            } else {
                compressionThreshold = 0;
            }

        } else {
            compressionThreshold = 0;
        }

    }

    /**
    * Take this filter out of service.
    */
    @Override
    public void destroy() {
        this.config = null;
    }

    /**
     * The <code>doFilter</code> method of the Filter is called by the container
     * each time a request/response pair is passed through the chain due
     * to a client request for a resource at the end of the chain.
     * The FilterChain passed into this method allows the Filter to pass on the
     * request and response to the next entity in the chain.<p>
     * This method first examines the request to check whether the client support
     * compression. <br>
     * It simply just pass the request and response if there is no support for
     * compression.<br>
     * If the compression support is available, it creates a
     * CompressionServletResponseWrapper object which compresses the content and
     * modifies the header if the content length is big enough.
     * It then invokes the next entity in the chain using the FilterChain object
     * (<code>chain.doFilter()</code>), <br>
     **/

    @Override
    public void doFilter ( ServletRequest request, ServletResponse response,
                           FilterChain chain ) throws IOException, ServletException {

        if (debug > 0) {
            System.out.println("@doFilter");
        }
        if (compressionThreshold == 0) {
```

```
                if (debug > 0) {
                    System.out.println("doFilter gets called, but compressionTreshold is set to 0 - no
compression");
                }
                chain.doFilter(request, response);
                return;
            }
            boolean supportCompression = false;
            if (request instanceof HttpServletRequest) {
                if (debug > 1) {
                    System.out.println("requestURI = " + ((HttpServletRequest)request).getRequestURI());
                }
                // Are we allowed to compress ?
                String s = ((HttpServletRequest)request).getParameter("gzip");
                if ("false".equals(s)) {
                    if (debug > 0) {
                        System.out.println("got parameter gzip=false --> don't compress, just chain
filter");
                    }
                    chain.doFilter(request, response);
                    return;
                }

                Enumeration<String> e =
                    ((HttpServletRequest)request).getHeaders("Accept-Encoding");
                while (e.hasMoreElements()) {
                    String name = e.nextElement();
                    if (name.indexOf("gzip") != -1) {
                        if (debug > 0) {
                            System.out.println("supports compression");
                        }
                        supportCompression = true;
                    } else {
                        if (debug > 0) {
                            System.out.println("no support for compresion");
                        }
                    }
                }
            }
            if (!supportCompression) {
                if (debug > 0) {
                    System.out.println("doFilter gets called wo compression");
                }
                chain.doFilter(request, response);
                return;
            }
```

```
            if (response instanceof HttpServletResponse) {
                CompressionServletResponseWrapper wrappedResponse =
                    new CompressionServletResponseWrapper((HttpServletResponse)response);
                wrappedResponse.setDebugLevel(debug);
                wrappedResponse.setCompressionThreshold(compressionThreshold);
                if (debug > 0) {
                    System.out.println("doFilter gets called with compression");
                }
                try {
                    chain.doFilter(request, wrappedResponse);
                } finally {
                    wrappedResponse.finishResponse();
                }
                return;
            }
        }
        /**
         * Set filter config
         * This function is equivalent to init. Required by Weblogic 6.1
         * @param filterConfig The filter configuration object
         */
        public void setFilterConfig(FilterConfig filterConfig) {
            init(filterConfig);
        }
        /**
         * Return filter config
         * Required by Weblogic 6.1
         */
        public FilterConfig getFilterConfig() {
            return config;
        }
}
```

```
        <filter>
            <filter-name>Compression Filter</filter-name>
            <filter-class>compressionFilters.CompressionFilter</filter-class>
            <init-param>
               <param-name>compressionThreshold</param-name>
               <param-value>10</param-value>
            </init-param>
            <init-param>
               <param-name>debug</param-name>
               <param-value>0</param-value>
            </init-param>
        </filter>
            <filter-mapping>
```

```
            <filter-name>Compression Filter</filter-name>
            <url-pattern>/CompressionTest</url-pattern>
        </filter-mapping>
```

```
package compressionFilters;

import java.io.IOException;
import java.util.Enumeration;

import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Very Simple test servlet to test compression filter
* @version $Id: CompressionFilterTestServlet.java 982412 2010-08-04 21:55:19Z markt $
 */

public class CompressionFilterTestServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        ServletOutputStream out = response.getOutputStream();
        response.setContentType("text/plain");
        Enumeration<String> e = request.getHeaders("Accept-Encoding");
        while (e.hasMoreElements()) {
            String name = e.nextElement();
            out.println(name);
            if (name.indexOf("gzip") != -1) {
                out.println("gzip supported -- able to compress");
            }
            else {
                out.println("gzip not supported");
            }
        }
        out.println("Compression Filter Test Servlet");
        out.close();
    }
}
```