欢迎大家进入就业班进行 JavaEE 培训

姓名:杜宏
电话:18600774790
QQ :22597927
EMAIL:duhong@itcast.cn

第一天 java 技术加强

Eclipse 的使用

- 工作空间(workspace)、工程(project)、工作组(working set)
- 在 eclipse 下 Java 程序的编写和运行，及 java 运行环境的配置。
- 快捷键的配置，常用快捷键：
  - 内容提示(Content Assist)：Alt + /
  - 快速修复(Quick Fix)：Ctrl + 1
  - 导包：ctrl + shift + O
  - 格式化代码块：ctrl + shift + F ---> 配置代码自动格式化
  - 添加(除去)块注释 Ctrl+Shift+/ (Ctrl+Shift+\)
  - 添加(除去)单行注释 Ctrl+/
- 移动代码：选中行 alt+上/下
- 删除行：ctrl+D

- 使用 Eclipse 的 Debug 功能
  - 查看类源代码：ctrl+鼠标左键/ F3
  - 前向后：Alt + 方向键(左、右)
  - 查看类继承关系：F4
  - F5(跳入)　F6(跳过)　　F7(跳出) debug 模式

Junit 测试

- 取代 main 方法快速测试程序
- @Test: 测试方法
- @Ignore: 被忽略的测试方法
- @Before: 在每个测试方法执行之前都要执行一次。
- @After: 在每个测试方法执行之后要执行一次。
- @BeforeClass: 所有测试开始之前运行 static
- @AfterClass: 所有测试结束之后运行 static
- 使用断言判断测试结果
  - assertEquals(expected, actual)
  - assertNull(object)
  - assertNotNull(object)
  - assertTrue(condition)
  - assertFalse(condition)

jdk1.5 新特性

泛型

泛型优点 1.安全　2.不需要强制转换　3.将运行阶段问题放到编译阶段解决

泛型:1.使用泛型规范集合，并且会对使用了泛型的集合 List　Map 进行遍历

1.集合中的泛型必须会使用

2.我们在自定义类上使用泛型

2.1 在类上定义泛型

在类上定义了一个泛型,对于在类上定义的泛型，我们在整个类内都可以使用.可以在方法上，可以在属性上，但是不能在静态方法中应用。

2.2 在方法上定义泛型

在任意的方法上都能声明泛型，但是泛型必须加在方法的返回值前。

3.泛型中的通配符 关于通配符使用

通配符是　?

? extends E　它代表的是 E 类型或 E 的子类

这个在 Collection 接口中的 addAll(Collection<? extends E> c)

? super E　　它代表的是 E 类型或 E 的父类

泛型的擦除

泛型只是在编译阶段有效果，当运行后它就失去作用。

枚举

- 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
- 创建枚举格式：

```
enum 枚举类型名称 {
    枚举对象 1 名称,
    枚举对象 2 名称,
     … ,
    枚举对象 n 名称;
}
```

- 枚举类具有如下特性：
  - 枚举类也是一种特殊形式的 Java 类。
  - 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。
  - 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（这点不难理解）。
  - 若枚举类只有一个枚举值，则可以当作单态设计模式使用。
- 枚举注意事项
  - 1：枚举默认就是 abstract 的，不可以实例化。
  - 2：枚举中可以拥有 abstract 抽像方法。
  - 3：枚举的所有成员，都默认是 public static final 类型的。且必须要第一行开始声明。必须在,（逗号）分开。
  - 4：所有的枚举，默认都是 java.lang.Enum。
  - 5：Enum 类不但是所有枚举的公共的基类，还是一个工具类。
  - 6：枚举中的构造方法必须是 private，可以重载。
- 枚举常用 API

  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()　得到枚举常量的名称
  - ordinal() 得到枚举常量的序号
  - valueof(Class enumClass, String name) 根据枚举常量的名称得到枚举对象
  - 自定义的枚举类
  - valueof(String name) 根据枚举常量名称得到枚举对象
  - 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

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：
  - import static java.lang.System.out;
  - import static java.util.Arrays.sort;
  - import static java.lang.Math.*;

静态导入它有弊端

如果一个类中存在同名的方法，在导入时就会出错。

静态导入的代码可读性太差。

自动拆箱与装箱

- 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：

```
List list = new ArrayList();
list.add(1);
int j = (Integer)list.get(0);
```

增强 for

- 引入增强 for 循环的原因：抛弃 Iterator
- for/in 语句的适用范围
  - 遍历**数组**
  - 遍历**实现** Iterable **接口**的集合类
- 语法格式：

```
for(变量类型 变量 ：需迭代的数组或集合){

}
```

增强 for 底层是使用迭代器实现的。

使用它的时候只能进行遍历操作，而不能像 List 中通过 get 方式可以对某个元素进行精确控制。

什么样的类可以被增强 for 操作？ 只要实现 Iterable 接口就可以。

这个接口代表是可以被增强 for 操作的。

动态参数

- 测试 JDK 中具有可变参数的类 Arrays.asList()方法。分别传多个参、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

声明 类型... 变量名

这个变量本质就是一个数组.
在传递参数时，可以传递多个值，也可以传递一个数组。
注意：动态后面不能有其它的参数，可以在前面声明。

Arrays 下的 asList 就是使用了动态参数，
它的作用是将数组变成 List 集合。
注意：由数组变成的集合不能改变集合长度。
如果数组中的元素是基本类型，那么它会将数组装入到集合中做为元素。
如果是引用类型，将数组中的元素做为集合的元素。

反射

- 什么是反射？
  - 剖析 Java 类中的各个组成部分映射成一个个 java 对象
  - 类 java.lang.Class
  - java.lang.reflect 包下
  - 构造方法 Constructor
  - 成员变量 Field
  - 方法 Method

- 反射用在哪里
  - 多用于框架和组件，写出复用性高的通用程序

➢ Class 的获得

- Java 中 java.lang.Class 类用于表示一个类的字节码(.class)文件

- 如何得到某个 class 文件对应的 class 对象
    - 已知类和对象的情况下
        - 类名.class
        - 对象.getClass()
    - 未知类和对象的情况下
        - Class.forName(“包名.类名”)

- Class 类代表某个类的字节码，并提供了加载字节码的方法：forName(“包名.类名”)
    - forName 方法用于加载类字节码到内存中，并封装成一个 Class 对象

Constructor 类

- **Constructor 类的实例对象代表类的一个构造方法**

- **得到某个类所有的构造方法**

```
Constructor [] constructors= Class.forName("java.lang.String").getConstructors();
```

- **得到指定的构造方法并调用**

```
Constructor constructor = Class.forName(“java.lang.String”).getConstructor(String.class);
String str = (String)constructor.newInstance(“abc”);
```

- **Class 类的 newInstance()方法用来调用类的默认构造方法**

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

Field 类

- Field 类代表某个类中的一个成员变量，并提供动态的访问权限
- Field 对象的获得
    - 得到所有的成员变量
        - Field[] fields = c.getFields(); // 取得所有 public 属性
        - Field[] fields = c.getDeclaredFields(); // 取得所有声明的属性
    - 得到指定的成员变量
        - Field name = c. getField("name");
        - Field name = c. getDeclaredField("name");
- 设置 Filed 变量是否可以访问
    - field. setAccessible(boolean);
- Field 变量值的读取、设置
    - field.get(obj)
    - filed.set(obj,value);

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：
  - char charAt(int index)
  - 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。

JDK1.7 新特性

二进制符号

你可用作二进制字符前加上 0b 来创建一个二进制类型。
int binary = 0b1001_1001;

在数字中使用下划线

int billion = 1_000_000_000;

对字符串进行 switch case

```
String availability = "available";
switch(availability) {
 case "available":
   //code
   break;

 case "unavailable":
   //code
```

```
    break;

  case "merged":
    //code

  default:
    //code
    break;
}
```

注意：在把字符串传进 Switch case 之前，别忘了检查字符串是否为 Null。

泛型实例创建过程中类型引用的简化

当声明你的对象的接口是，你肯定是不想重复指明泛型类型

看看下面的这种写法多好呀…

```
Map<String,String> hello = new Map<>();
```

对资源的自动回收管理

下面的代码看起来有点麻烦 … 不是吗

```
BufferedReader br = new BufferedReader(new FileReader(path));
try {
  return br.readLine();
} finally {
  br.close();
}
```

相信你一定会喜欢下面这种写法

```
try (BufferedReader br = new BufferedReader(new FileReader(path)) {
  return br.readLine();
}
```

一个 catch 里捕捉多个异常类型

```
try {

Here comes your code....
```

```
}

catch(IOException | NullPointerException | .....) {

}
```

第二天 xml 与 xml 约束

XML

什么是 XML(Extensible Markup Language)

- XML 指可扩展标记语言
- XML 是一种标记语言，很类似 HTML
- XML 的设计宗旨是传输数据，而非显示数据
- XML 标签没有被预定义
- 使用约束的 XML 文档设计具有自我描述性。
- XML 是 W3C 的推荐标准

XML 企业端应用

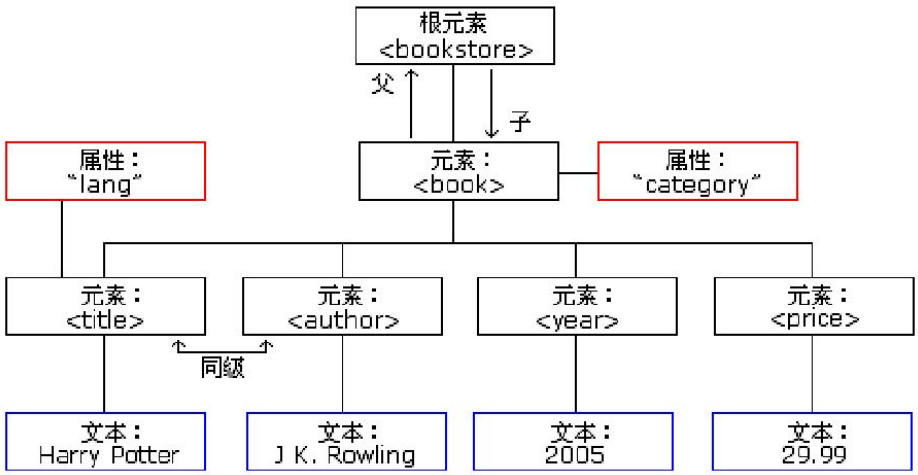
- 存储和传输复杂的关系模型数据
- 在软件系统中，作为配置文件使用

XML 与 HTML 的主要差异

- XML 不是 HTML 的替代。
- XML 和 HTML 为不同的目的而设计：
- XML 被设计为传输和存储数据，其焦点是数据的内容。
- HTML 被设计用来显示数据，其焦点是数据的外观。
- HTML 旨在显示信息，而 XML 旨在传输信息。

XML 树形结构

XML 文档必须包含*根元素*。该元素是所有其他元素的父元素。XML 文档中的元素形成了一棵文档树。这棵树从根部开始，并扩展到树的最底端




```
<bookstore>
    <book category="COOKING">
        <title lang="en">Everyday Italian</title>
        <author>Giada De Laurentiis</author>
        <year>2005</year>
        <price>30.00</price>
    </book>
    <book category="WEB">
        <title lang="en">Learning XML</title>
        <author>Erik T. Ray</author>
        <year>2003</year>
        <price>39.95</price>
    </book>
</bookstore>
```

XML 的组成部分

- 文档声明

  在编写 XML 文档时，需要先使用文档声明，声明 XML 文档的类型。

  最简单的声明语法：

  <?xml version="1.0" ?>

  用 encoding 属性说明文档的字符编码：

  <?xml version="1.0" encoding="GB2312" ?>

  用 standalone 属性说明文档是否独立：

  <?xml version="1.0" encoding="GB2312" standalone="yes" ?>

- 元素是　XML 以及 HTML 文档的*主要构建模块*

  XML 元素指 XML 文件中出现的标签，一个标签分为开始标签和结束标签，一个标签有如下几种书写形式，例如：

  包含标签体：<a>www.itcast.cn

不含标签体的：<a>, 简写为：<a/>

一个标签中也可以嵌套若干子标签。但所有标签必须合理的嵌套，绝对不允许交叉嵌套 ，例如：

```
<a>welcome to <b>www.it315.org</a></b>
```

格式良好的 XML 文档必须有且仅有一个根标签，其它标签都是这个根标签的子孙标签。

对于 XML 标签中出现的所有空格和换行，XML 解析程序都会当作标签内容进行处理。例如：下面两段内容的意义是不一样的

第一段：

```
<网址>www.itcast.cn</网址>
```

第二段：

```
<网址>
    www.itcast.cn
</网址>
```

由于在 XML 中，空格和换行都作为原始内容被处理，所以，在编写 XML 文件时，使用换行和缩进等方式来让原文件中的内容清晰可读的“良好”书写习惯可能要被迫改变

XML 元素必须遵循以下命名规则：

- 区分大小写名称可以含字母、数字以及其他的字符
- 名称不能以数字或者标点符号开始
- 名称不能以字符 “xml”（或者 XML、Xml）开始
- 名称不能包含空格
- 避免 ":" 字符。冒号会被转换为命名空间来使用（稍后介绍）。

- 属性　　可提供*有关元素的额外信息*

一个标签可以有多个属性，每个属性都有它自己的名称和取值，例如：

```
<input type=”text”>
```

属性值一定要用双引号（"）或单引号（'）引起来

定义属性必须遵循与标签相同的命名规范

多学一招：在 XML 技术中，标签属性所代表的信息，也可以被改成用子元素的形式来描述，例如：

```
<input>
    <name>text</name>
</input>
```

避免 XML 属性？因使用属性而引起的一些问题：

- 属性无法包含多个值（子元素可以）
- 属性无法描述树结构（子元素可以）
- 属性不易扩展（为未来的变化）
- 属性难以阅读和维护

请尽量使用元素来描述数据。而仅仅使用属性来提供与数据无关的信息

- 注释

Xml 文件中的注释采用：“<!--注释-->”格式。

注意：

XML 声明之前不能有注释

注释不能嵌套，例如：

```
<!--大段注释
……
    <!--局部注释-->
……
-->
```

- CDATA 区 、特殊字符

在编写 XML 文件时，有些内容可能不想让解析引擎解析执行，而是当作原始内容处理。

遇到此种情况，可以把这些内容放在 CDATA 区里，对于 CDATA 区域内的内容，XML 解析程序不会处理，而是直接原封不动的输出。

语法：<![CDATA[内容]]>

```
<![CDATA[
    <itcast>
        <br/>
    </itcast>
]]>
```

对于一些单个字符，若想显示其原始样式，也可以使用转义的形式予以处理。

特殊字符	替代符号
&	&amp;
<	&lt;
>	&gt;
"	&quot;
'	&apos;

- 处理指令（processing instruction）

处理指令，简称 PI （processing instruction）。处理指令用来指挥解析引擎如何解析 XML 文档内容。

例如，在 XML 文档中可以使用 xml-stylesheet 指令，通知 XML 解析引擎，应用 css 文件显示 xml 文档内容。 <?xml-stylesheet type="text/css" href="1.css"?>

处理指令必须以“<?”作为开头，以“?>”作为结尾，XML 声明语句就是最常见的一种处理指令。

XML 语法总结

XML 的语法规则很简单，且很有逻辑。这些规则很容易学习，也很容易使用。

- 所有 XML 元素都须有关闭标签

- XML 标签对大小写敏感
- XML 必须正确地嵌套
- XML 文档必须有根元素
- XML 的属性值须加引号
- 特殊字符必须转义
- XML 中的空格会被保留

XML 约束介绍

- 什么是 XML 约束
  - 在 XML 技术里，可以编写一个文档来约束一个 XML 文档的书写规范，这称之为 XML 约束。
- 为什么需要 XML 约束
- 常用的约束技术
  - XML DTD
  - XML Schema

DTD

什么是 DTD

DTD（文档类型定义）的作用是定义 XML 文档的合法构建模块。它使用一系列的合法元素来定义文档结构. DTD 可被成行地声明于 XML 文档中，也可作为一个外部引用。

拥有正确语法的 XML 被称为“格式良好”的 XML。通过某个 DTD 进行了验证的 XML 是“合法”的 XML。

为什么使用 DTD？

通过 DTD，您的每一个 XML 文件均可携带一个有关其自身格式的描述。通过 DTD，独立的团体可一致地使用某个标准的 DTD 来交换数据。而您的应用程序也可使用某个标准的 DTD 来验证从外部接收到的数据。您还可以使用 DTD 来验证您自身的数据。

DTD 示例

Xml 文件内容

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE books SYSTEM "books.dtd">
<books>
    <book lang="en">
```

```
        <num></num>
        <author>tom</author>
        <price>99</price>
    </book>
    <book lang="zh">
        <name>thinking in c++</name>
        <author>james</author>
        <price>88</price>
    </book>

    <book lang="ch">
        <name>thinking in c++</name>
        <author>james</author>
        <price>88</price>
    </book>
</books>
```

Dtd 文件内容

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT books (book+) >
<!ELEMENT book ( (name|num),author,price)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT author (#PCDATA) >
<!ELEMENT price (#PCDATA) >
<!ELEMENT num EMPTY >


<!ATTLIST book
    lang ID #REQUIRED
>
```

DTD 校验

1. 使用 eclipse 工具自动校验
2. 使用 js 校验

```
<script type="text/javascript">
    var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")
    xmlDoc.async = "false";
    xmlDoc.validateOnParse = "true";
    xmlDoc.load("bookstore.xml");

    document.write("<br>Error Code: ");
```

```
    document.write(xmlDoc.parseError.errorCode);
    document.write("<br>Error Reason: ");
    document.write(xmlDoc.parseError.reason);
    document.write("<br>Error Line: ");
    document.write(xmlDoc.parseError.line);
</script>
```

将 DTD 与 XML 文档关联三种方式

DTD 约束即可以作为一个单独的文件编写，也可以在 XML 文件内编写

- 使用内部 DTD
- 使用外部 DTD　SYSTEM
- 使用公共 DTD　PUBLIC

➢ 内部 DTD

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE 书架 [
 <!ELEMENT 书架 (书+)>
 <!ELEMENT 书 (书名,作者,售价)>
 <!ELEMENT 书名 (#PCDATA)>
 <!ELEMENT 作者 (#PCDATA)>
 <!ELEMENT 售价 (#PCDATA)>
]>
<书架>
 <书>
      <书名>Java 就业培训教程</书名>
      <作者>张孝祥</作者>
      <售价>39.00 元</售价>
 </书>
 ...
</书架>
```

➢ 外部 DTD（文件的后缀名是 dtd）

XML 文件使用 DOCTYPE 声明语句来指明它所遵循的 DTD 文件，DOCTYPE 声明语句有两种形式：

当引用的文件在本地时（外部 DTD），采用如下方式：

<!DOCTYPE 文档根结点 SYSTEM "DTD 文件的 URL">

例如： <!DOCTYPE 书架 SYSTEM “book.dtd”>。在 xml 文件中手写一下。

当引用的文件是一个公共的文件时（公共 DTD）采用如下方式：

<!DOCTYPE 文档根结点 PUBLIC "DTD 名称" "DTD 文件的 URL">

例如：<!DOCTYPE web-app PUBLIC

"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

DTD 元素

<!ELEMENT 元素名称 元素内容声明>

要点：包含符号和数据类型两部分。

符号	符号类型	描述	示例
?	问号	表示该对象可以出现，但只能出现一次	（菜鸟?）
*	星号	表示该对象允许出现任意多次，也可以是零次	（爱好*）
+	加号	表示该对象最少出现一次，可以出现多次	（成员+）
0	括号	用来给元素分组	（古龙|金庸|梁羽生），（王朔|余杰），毛毛
|	竖条	表明在列出的对象中选择一个	（男人|女人）
,	逗号	表示对象必须按指定的顺序出现	（西瓜,苹果,香蕉）

元素内容类型

内容	解释
#PCDATA	表明该元素可以包含任何字符数据，但是不能在其中包含任何子元素。 假设我们定义元素学员： <!ELEMENT 学员 #PCDATA> 则下面的实例是正确的： <学员>努力学习是好学员</学员> 而下面的实例就是错误的： <学员>努力<B>学习<B>是好学员</学员> 因为在其中包含了子元素<B>。 一般如果定义元素的CONTENT为#PCDATA，最好在其中只加入纯文本字符数据
EMPTY	如果一个元素的CONTENT被声明为EMPTY的话，表示该元素不能包含任何子元素和文本，仅可以使用属性。
ANY	表示该元素中可以包含任何在DTD中定义的元素内容
其他类型	最通常的情况是一个元素本身是由其他元素的集合构成的

DTD 属性

<!ATTLIST 商品
    类别 CDATA #REQUIRED
    颜色 CDATA #IMPLIED
>
#REQUIRED 这个代表属性必须有。

DTD 实体

引用实体
<!DOCTYPE persons [
    <!ENTITY 实体名 “实体值”>
]>
引用时 &实体名;

```
<!DOCTYPE persons [
    <!ENTITY company SYSTEM "entity_content.xml" >  //导入外部实体
]>
```

```
参数实体（它只能应用在 dtd 中）
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % basic "name,age">
<!ENTITY company "传智播客">
<!ELEMENT user (programer,manager)>
<!ELEMENT programer (%basic;,birthday)>
<!ELEMENT manager (%basic;,sal,company)>
```

SCHEMA

Schema 介绍

- 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

Schema 名称空间

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                    targetNamespace="http://www. itcast.cn"
                    elementFormDefault="qualified"
                    attributeFormDefault="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 元素用于指定局部元素是否受到该 schema 指定 targetNamespace 所指定的名称空间限定
- attributeFormDefault 元素用于指定局部属性是否受到该 schema 指定 targetNamespace 所指定的名称空间限定

- 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。
- 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。

Xml 解析

- Xml 解析概述

- 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 组织推荐的解析 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 解析开发包
  - Jaxp(sun)、Jdom、dom4j

➢ Sax 解析与 dom 解析的区别

- DOM 支持回写
  - 会将整个 XML 载入内存，以树形结构方式存储
  - 一个 300KB 的 XML 文档可以导致 RAM 内存或者虚拟内存中的 3，000，000KB 的 DOM 树型结构
  - XML 比较复杂的时候，或者当你需要随机处理文档中数据的时候不建议使用
- SAX
  - 相比 DOM 是一种更为轻量级的方案
  - 采用串行方法读取 --- 逐行读取
  - 编程较为复杂
  - 无法修改 XML 数据

➢ 选择 DOM 还是 SAX，这取决于几个因素

- 应用程序的目的：如果必须对数据进行更改，并且作为 XML 将它输出，则在大多数情况下，使用 DOM
- 数据的数量：对于大文件，SAX 是更好的选择
- 将如何使用数据：如果实际上只使用一小部分数据，则使用 SAX 将数据抽取到应用程序中，这种方法更好些
- 需要速度：通常，SAX 实现比 DOM 实现快

Dom 解析

- 解析器工厂类 DocumentBuilderFactory
  - DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
- 解析器类 DocumentBuilder
  - DocumentBuilder db = dbf.newDocumentBuilder();
- 解析生成 Document 对象
  - Document doc = db.parse("message.xml");
- 通过 Document 对象查询节点
  - document.getElementById 返回 Node 对象 --- 必须文档元素有 ID 属性
  - document.getElementsByTagName 返回 NodeList 对象

➢ 查询节点 API

通过 Document 对象查询节点

- document.getElementById　返回 Node 对象 --- 必须文档元素有 ID 属性
- document.getElementsByTagName　返回 NodeList 对象

➢ 什么是节点：元素、属性、文本、实体、注释

Node 子接口： Element、Attr、Text、Document

➢ NodeList API

节点列表类 NodeList 就是代表了一个包含一个或者多个 Node 的列表,可以简单的把它看成一个 Node 的数组

- 常用方法
  - getLength()：返回列表的长度。
    - ArrayList size
  - item(int)：返回指定位置的 Node 对象
    - ArrayList get(index)

➢ Node API

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

- getAttribute(String)：返回标签中给定属性的值(元素)
- getAttributeNode(name)：返回指定名称的属性节点（元素）
- getNodeName()：返回节点的名称（元素-- > 标签名）
- getNodeType()：返回节点的类型
- getNodeValue()：返回节点的值
- getChildNodes()：返回这个节点的所有子节点列表
- getFirstChild()：返回这个节点的第一个子节点
- getParentNode()：返回这个节点的父节点对象
- appendChild(org.w3c.dom.Node)：为这个节点添加一个子节点，并放在所有子节点的最后，如果这个子节点已经存在，则先把它删掉再添加进去
- removeChild(org.w3c.dom.Node)：删除给定的子节点对象
- replaceChild(org.w3c.dom.Node new，org.w3c.dom.Node old)：用一个新的 Node 对象代替给定的子节点对象
- getNextSibling()：返回在 DOM 树中这个节点的下一个兄弟节点
- getPreviousSibling()返回在 DOM 树中这个节点的前一个兄弟节点

➢ dom 回写

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。

➢ Dom CRUD 示例

Sax 解析的

根据 element 可以获得元素名称与属性内容
根据 character 可以获取文本内容.

- SAX 是事件驱动的 XML 处理方法
- 它是基于事件驱动的
- startElement() 回调在每次 SAX 解析器遇到元素的起始标记时被调用
- characters() 回调为字符数据所调用
- endElement() 为元素的结束标记所调用
- DefaultHandler 类（在 org.xml.sax.helpers 软件包中）来实现所有这些回调，并提供所有回调方法默认的空实现

➢ Sax 解析步骤

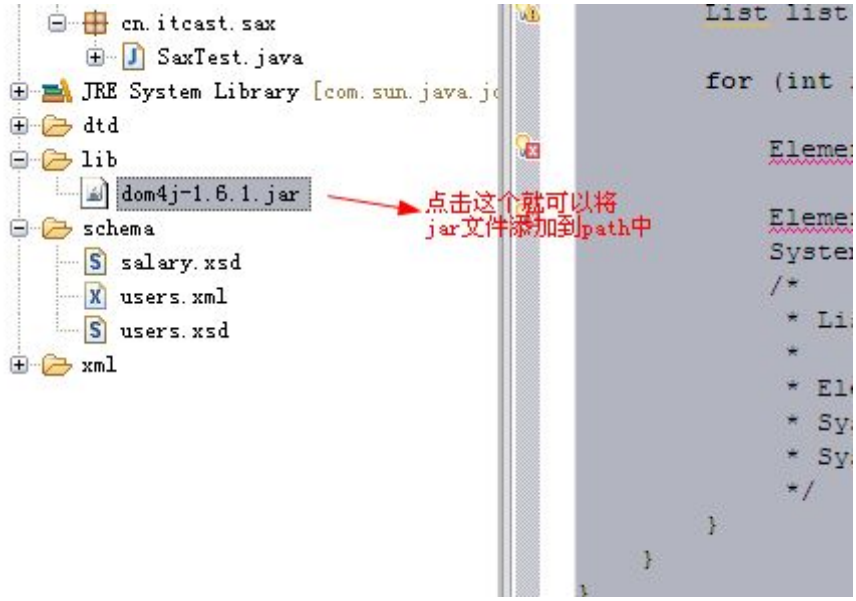
- 使用 SAXParserFactory 创建 SAX 解析工厂
  - SAXParserFactory spf = SAXParserFactory.newInstance();
- 通过 SAX 解析工厂得到解析器对象
  - SAXParser sp = spf.newSAXParser();
- 通过解析器对象得到一个 XML 的读取器
  - XMLReader xmlReader = sp.getXMLReader();
- 设置读取器的事件处理器
  - xmlReader.setContentHandler(new XMLContentHandler());
- 解析 xml 文件
  - xmlReader.parse("book.xml");

Dom4j 解析

Dom4j 介绍

- 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

Dom4j 导包



Dom4j 常用 API

- DOM4j 中，获得 Document 对象的方式有三种：

1.读取 XML 文件,获得 document 对象 （解析）

```
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. （添加--创建一个 xml 片段）

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

3.主动创建 document 对象. （创建一个空文档对象 --- 生成）

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

- DOM4J 常用 API

节点对象操作 API

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. root.elements();

```
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());
```

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”);
//getValue()
```

2.取得属性的文字

```
String text=attribute.getText(); === getValue();
```

3.删除某属性

```
Attribute attribute=root.attribute("size");
root.remove(attribute);
```

4.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);
}
```

5.设置某节点的属性和文字.

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

6.设置属性的文字

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

Dom4j CRUD

第三天 web 服务器与 HTTP 协议

Web 服务器

- WEB，在英语中 web 即表示网页的意思，它用于表示 Internet 主机上供外界访问的资源。
- Internet 上供外界访问的 Web 资源分为：
  - 静态 web 资源（如 html 页面）：指 web 页面中供人们浏览的数据始终是不变。
  - 动态 web 资源：指 web 页面中供人们浏览的数据是由程序产生的，不同时间点访问 web 页面看到的内容各不相同。
- 静态 web 资源开发技术
  - Html
- 常用动态 web 资源开发技术：
  - JSP/Servlet、ASP、PHP 等 ruby python
  - 在 Java 中，动态 web 资源开发技术统称为 Javaweb，我们课程的重点也是教大家如何使用 Java 技术开发动态的 web 资源，即动态 web 页面。

但是我们做 java 开发，不是做网页。

网络上的资源分为两种

早期:静态页面 html 实现。 观看

现在:动态页面 php asp jsp 交互.

lamp =linux +apache+ mysql+php----->个人网关或小型企业首选

asp 现在没人用，但是网络上遗留下来的比较多。miscrosoft 的技术

.net 技术。

jsp--->java 去做网页所使用的技术。jsp 本质上就是 servlet

使用 jsp 开发成本高。

BS====>浏览器+服务器 只要有浏览器就可以

CS----->客户端+服务器. 必须的在客户端安装程序.

现在基本上开发的都是 BS 程序

BS 怎样通信:

必须有请求有响应。

有一次请求就应该具有一次响应，它们是成对出现的。

服务器介绍

大型服务器：websphere(IBM),weblogic(Oracle)　J2EE 容器　—

支持 EJB (EnterPrice Java Bean (企业级的 javabean)) – Spring

weblogic BEA 公司产品，被 Oracle 收购，全面支持 JavaEE 规范，收费软件，企业中非常主流的服务器 -------- 网络上文档非常全面

WebSphere 文档非常少，IBM 公司产品，价格昂贵，全面支持 JavaEE 规范

Tomcat- apache，开源的。Servlet 容器。

tomcat 开源小型 web 服务器 ，完全免费，主要用于中小型 web 项目，只支持 Servlet 和 JSP 等少量 javaee 规范 ，Apache 公司 jakarta 一个子项目

Jboss – hibernate 公司开发。不是开源免费。J2EE 容器

Tomcat 安装

注意路径中不要包含空格与中文。

- 安装步骤

1、tomcat.apache.org 下载 tomcat 安装程序

tomcat6 安装程序 ---- zip 免安装版

2、解压 tomcat

3、配置环境变量 JAVA_HOME 指向 JDK 安装目录　D:\Program Files\Java\jdk1.6.0_21

*CATALINA_HOME 指定 tomcat 安装目录

4、双击 tomcat/bin/startup.bat

5、在浏览器中 输入 localhost:8080 访问 tomcat 主页了

- 注意问题：

启动黑色不能关闭

1、CATALINA_HOME 指定 tomcat 安装位置 --- 可以不配置

2、JAVA_HOME 指定 JDK 安装目录，不要配置 bin 目录，不要在结尾加;

3、端口被占用

启动 cmd

netstat -ano 查看占用端口进程 pid

任务管理器 查看---选择列 显示 pid -- 根据 pid 结束进程

* 有些进程无法关系（系统服务 --- 必须结束服务）win7 自带 World wide web publish

IIS 服务 默认占用端口 80
* xp 安装 apache 服务器后，会占用 80 端口 ，关闭 apache 服务
通过运行 services.msc 打开服务窗口 关闭相应服务

tomcatc 目录结构

-----bin 它里面装入的是可执行的命令 如 startup.bat
-----conf 它里面是一个相关的配置文件，我们可以在里面进行例如端口，用户信息的配置

```
<Connector port="80" protocol="HTTP/1.1"
    connectionTimeout="20000"
    redirectPort="8443" />
```

-----lib tomcat 类库。
-----logs tomcat 日志文件
-----temp 临时文件
-----webapps 它里面放的是的 web site(web 项目)
-----work 存放的是页面(例如 jsp)转换成的.class 文件。

创建网站，将网站发布到 tomcat 服务器上

创建网站根目录

在根目录下 创建静态 web 资源和动态 web 资源

```
Web site
  ---- *.html *.css *.js 图片 音频 视频 、*.jsp
  ---- WEB-INF 目录 存放 java 程序和配置文件
          --- classes 存放.class 文件
          --- lib 存放.jar 文件
          --- web.xml 网站核心配置文件
```

*** 如果静态网站可以不存在 WEB-INF 目录的 ，WEB-INF 目录，客户端无法直接访问（在服务器内存通过程序访问）

将网站发布到 tomcat ----------虚拟目录映射

虚似目录的映射方式有三种

1.在开发中应用的比较多　直接在 webapps 下创建一个自己的 web site 就可以.

步骤 1.在 webapps 下创建一个 myweb 目录

2.在 myweb 下创建 WEB-INF 目录，在这个目录下创建 web.xml

3.将 web.xml 文件中的 xml 声明与根元素声明在其它的 web site 中 copy 过来。

4.在 myweb 下创建一个 index.html 文件

5.启动 tomcat

6.在浏览器中输入　http://localhost/myweb/index.html

以下两种方式，可以将 web site 不放置在 tomcat/webapps 下，可以任意放置

2.在 server.xml 文件中进行配置

<Context path="/abc"　docBase="C:\myweb1"/>
</Host>

在 Host 结束前配置

path：它是一个虚拟路径，是我们在浏览器中输入的路径
docBase：它是我们 web sit 的真实路径

http://localhost/abc/index.html

3.不在 server.xml 文件中配置

而是直接创建一个 abc.xml 文件
在这个 xml 文件中写
<Context path=""　docBase="C:\myweb1"/>

将这个文件放入 conf\Catalina\localhost

http://localhost/abc/index.html

生成 war 文件

war 文件是 web 项目的压缩文件。
要想生成，先将要压缩的内容压缩成 zip 文件，
然后将后缀改成 war 就可以，
war 文件可以直接在服务器上访问。

关于 tomcat-manager

可以在 conf/tomcat-users.xml 中进行用户信息添加

```
<role rolename="manager"/>
    <user username="xxx" password="xx" roles="manager"/>
```

这样就添加了一个用户

注意，用户权限要是比较大的话，会出现安全问题.

虚拟主机

做自己的一个 http://www.baidu.com

1.访问一个网站的过程

http://www.baidu.com

http 协议
www 服务器
.baidu.com 域名 IP

步骤

1.上网将 baidu 首页下载下来

2.做一个自己的 web site 首页就是下载下来的页面。
别忘记创建 WEB-INF 在它下面的 web.xml 文件中

```
<welcome-file-list>
  <welcome-file>index.html</welcome-file>
</welcome-file-list>
```

这句话的作用是默认访问页面是 index.html

3.在 tomcat 中的 conf 文件夹下的 server.xml 中修改

```
<Host name="www.baidu.com"    appBase="c:\baidu"
            unpackWARs="true" autoDeploy="true"
            xmlValidation="false" xmlNamespaceAware="false">


    <Context path="" docBase="c:\baidu"/>
</Host>
```

4.在 windows/system32/drivers/etc/hosts 中添加

127.0.0.1　　www.baidu.com

目的是当访问 www.baidu.com 时其实访问的是本机。

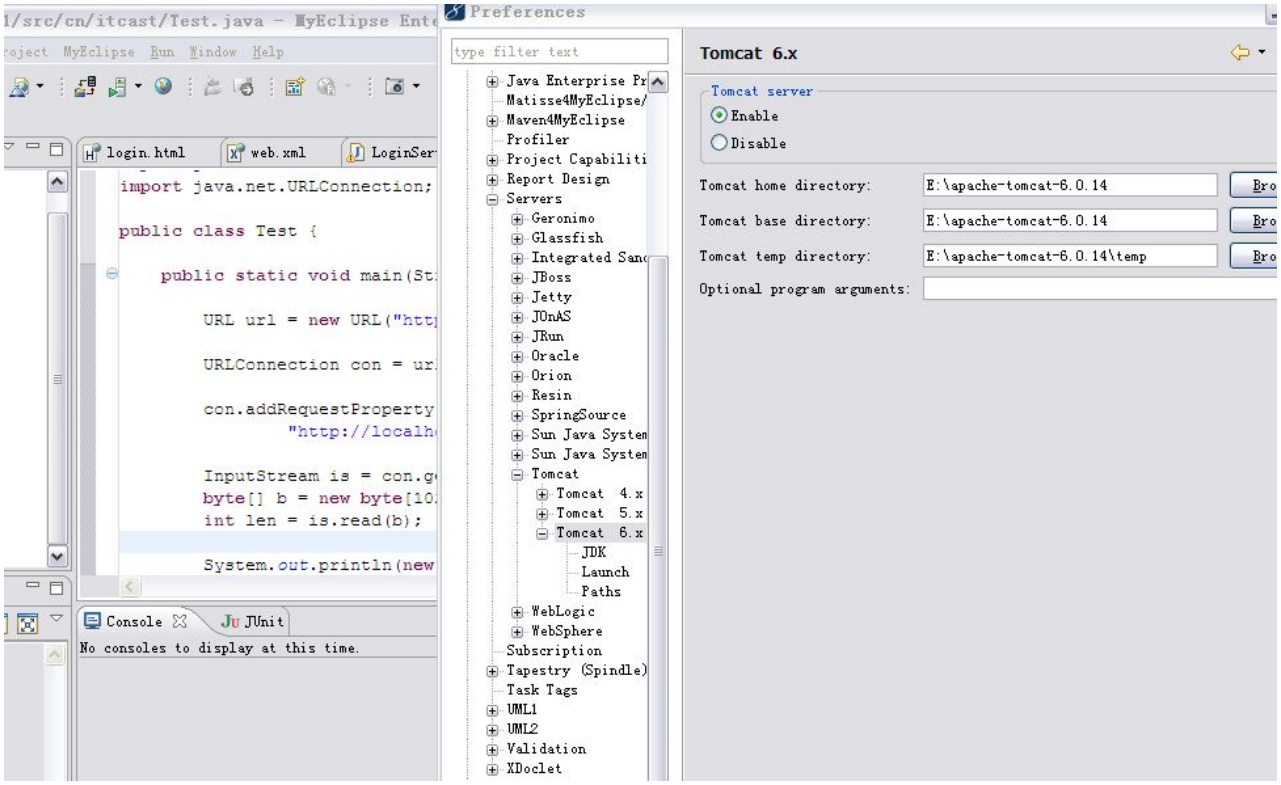
5.打开浏览器在地址栏中输入 www.baidu.com
这时其时访问的是我们自己
web site 中的页面。

使用 myeclipse 创建 web project 与 tomcat 集成

我们在 myeclipse 中创建 web project 有一个 WebRoot 目录。
但是我们发布到 tomcat 中没有这个，它其时就是我们工程的名称.

步骤

1.创建 web 工程
2.在 eclipse 中配置 tomcat 服务器
window/属性/myeclipse/service 中配置自己的 tomcat 目录.
注意到 tomcat 根目录就可以了。不要到 bin 中。
如果不好使用，看一些 jdk 是否配置.



3. 将 webproject 部署到 tomcat 中

HTTP 协议

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。

示例 1

telnet 怎样使用

1.telnet localhost 8080

2 ctrl+]

3.按回车

注意 在里面写错的内容不能修改

GET /index.html HTTP/1.1
host:localhost

4.要敲两次回车

HTTP/1.0 版本只能保持一次会话
HTTP/1.1 版本可能保持多次会话.

是根据 telnet 得到的响应信息

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
ETag: W/"7347-1184876416000"
Last-Modified: Thu, 19 Jul 2007 20:20:16 GMT
Content-Type: text/html
Content-Length: 7347
Date: Thu, 25 Apr 2013 08:06:53 GMT
Connection: close

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>Apache Tomcat</title>
<style type="text/css">
.....

示例 2

是根据 httpwatch 得到的请求信息与响应信息

请求
GET / HTTP/1.1
Accept: application/x-shockwave-flash, image/gif, image/jpeg, image/pjpeg, image/pjpeg, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Accept-Language: zh-cn
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)
Accept-Encoding: gzip, deflate
Host: localhost

Connection: Keep-Alive

响应

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
ETag: W/"7347-1184876416000"
Last-Modified: Thu, 19 Jul 2007 20:20:16 GMT
Content-Type: text/html
Content-Length: 7347
Date: Thu, 25 Apr 2013 08:12:57 GMT

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang

请求信息详解

GET /books/java.html HTTP/1.1 ---------->请求行
Get 是请求方式 /books/java.html 请求资源 HTTp/1.1 协议版本

POST 与 GET 的区别

1.什么样是 GET 请求 1)直接在地址栏输入 2.超连接 <a> 3.form 表单中 method=get

什么样是 POSt 请求 form 表单中 method=POST

2.以 get 方式提交请求时，在请求行中会将提交信息直接带过去

格式 /day03_1/login?username=tom&password=123

以 post 方式提交时，信息会在正文中。

POST /day03_1/login HTTP/1.1
Accept: application/x-shockwave-flash, image/gif, image/jpeg, image/pjpeg, image/pjpeg, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, */*
Referer: http://localhost/day03_1/login.html
Accept-Language: zh-cn
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
Host: localhost

Content-Length: 25
Connection: Keep-Alive
Cache-Control: no-cache

username=tom&password=123

3. get 方式最多能提交 1kb
post 可以提交大数据，做上传时必须是 post

Accept: */*　　允许访问 mime 类型,类型都在 tomcat 的 conf/web.xml 文件中定义了。
这个需要知道，因为做下载时要知道 mime 类型

Accept-Language: en-us 客户端的语言
Connection: Keep-Alive 持续连接
Host: localhost　客户端访问资源
Referer: http://localhost/links.asp (重点)　防盗链。
User-Agent: Mozilla/4.0　　得到浏览器版本 避免兼容问题
Accept-Charset: ISO-8859-1　客户端字符编码集
Accept-Encoding: gzip, deflate　　gzip 是压缩编码.
If-Modified-Since: Tue, 11 Jul 2000 18:23:51 GMT　与 Last-MOdified 一起可以控制缓存。
Date: Tue, 11 Jul 2000 18:23:51 GMT

示例 1

防盗链程序
referer.htm 页面

```
    <body>
        <a href="referer">referer</a>
    </body>
```

RefererServlet 类似

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

       String msg = request.getHeader("Referer");

       if (msg != null &&
"http://localhost/day03_1/referer.html".equals(msg)) {

          // 如果你是正常访问,我们给其一个友好信息
          response.getWriter().write("hello");
       } else {
          // 如果是盗链过来的，对不。骂它一句
```

```
            response.getWriter().write("fuck...");
        }
    }

怎样破解
    URL url = new URL("http://localhost/day03_1/referer"); //得到一个url

        URLConnection con = url.openConnection();   //访问这个url，并获得连接对象

        con.addRequestProperty("Referer",
                "http://localhost/day03_1/referer.html");

        InputStream is = con.getInputStream(); // 读取服务器返回的信息.
        byte[] b = new byte[1024];
        int len = is.read(b);

        System.out.println(new String(b, 0, len));
```

http 协议响应

HTTP/1.1 200 OK 响应状态行
HTTP/1.1 200 OK

1xx 什么都没做直接返回
2xx 成功返回
3xx 做了一些事情，没有全部完成。
4xx 客户端错误
5xx 服务器错误

200 正确
302 重定向
304 页面没有改变
404 未找到页面
500 服务器出错.

Location: http://www.it315.org/index.jsp 响应路径(重点)+302

Server:apache tomcat

Content-Encoding: gzip　响应编码　gzip 压缩

Content-Length: 80　响应长度

Content-Language: zh-cn　响应语言

Content-Type: text/html; charset=GB2312　响应字符编码

Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT　要与请求中的 If-Modified-Since 处理缓存

Refresh: 1;url=http://www.it315.org　自动跳转

Content-Disposition: attachment; filename=aaa.zip (重要) 文件的下载

//下面三个是禁用浏览缓存

Expires: -1

Cache-Control: no-cache

Pragma: no-cache

Connection: close/Keep-Alive

Date: Tue, 11 Jul 2000 18:23:51 GMT

重点

今天可以讲

Location: http://www.it315.org/index.jsp　响应路径(重点)+302

Last-Modified: Tue, 11 Jul 2000 18:23:51 GMT　要与请求中的 If-Modified-Since 处理缓存

Refresh: 1;url=http://www.it315.org　自动跳转

我们在得到响应信息，经常得到的是压缩后的。

这种操作

1.服务器配置方式

tomcat 配置实现压缩

80 端口没有配置　00:00:00.000 0.228　7553　GET200 text/html http://localhost/

8080 端口配置　00:00:00.000 0.027　2715　GET200 text/html http://localhost:8080/

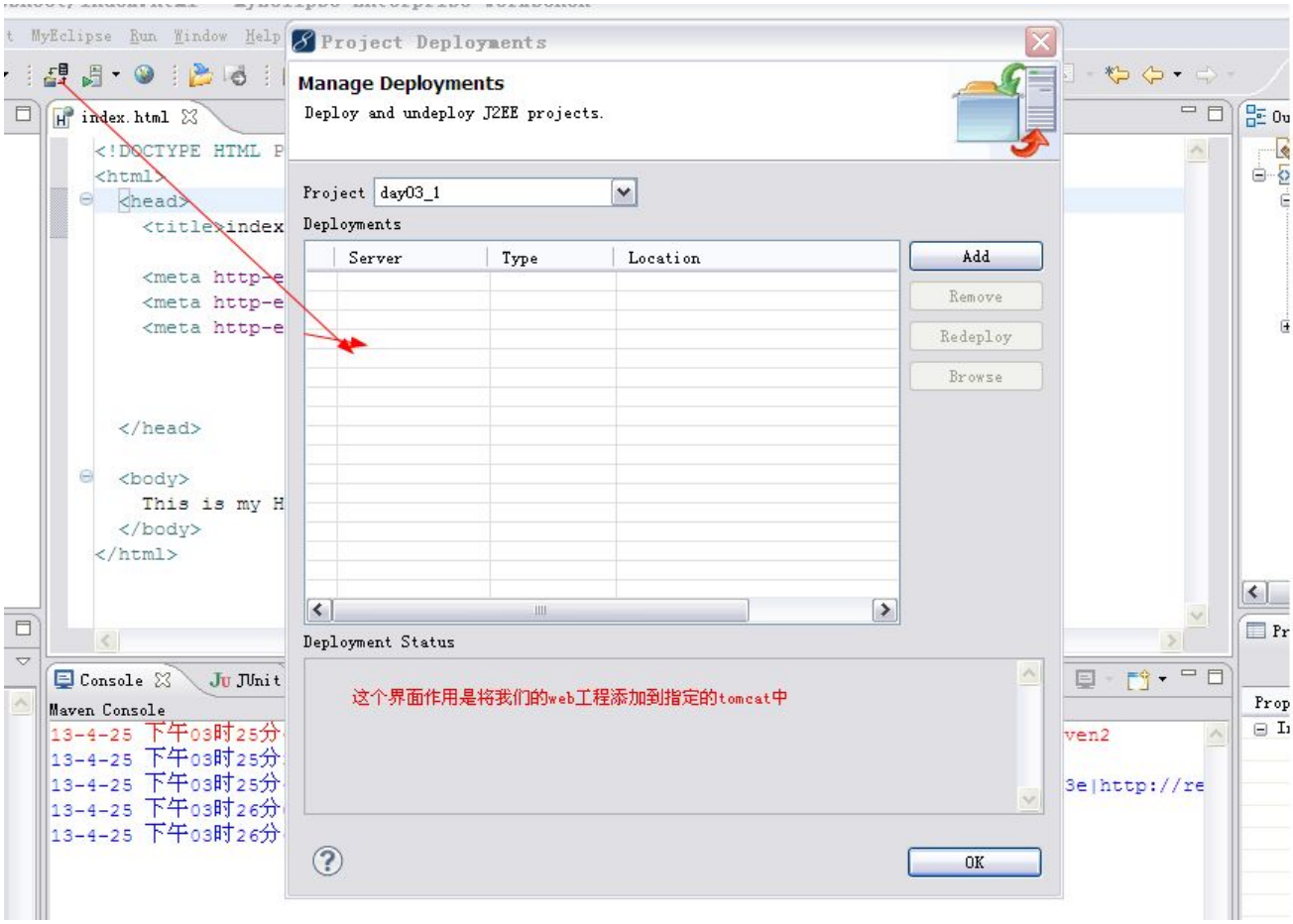
<Connector port="8080" protocol="HTTP/1.1"
connectionTimeout="20000"
redirectPort="8443" compressableMimeType="text/html,text/xml,text/plain" compression="on"/>

2.通过我们编程实现.(后面会讲)

后面会讲

Content-Disposition: attachment; filename=aaa.zip (重要) 文件的下载

//下面三个是禁用浏览缓存
Expires: -1
Cache-Control: no-cache
Pragma: no-cache



4.启动服务器

5.在浏览器中访问 web 资源.

第四天 Web 之 Servlet

Servlet

Servlet 介绍

- Servlet 是 sun 公司提供的一门用于开发动态 web 资源的技术
- Servlet 技术基于 Request-Response 编程模型
- Sun 公司在其 API 中提供了一个 servlet 接口，用户若想要开发一个动态 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

Servlet 创建步骤

- 继承 javax.servlet.http.HttpServlet
- web.xml 配置 Servlet 的虚拟路径
- 覆盖 doGet、doPost

示例 Servlet 类编写

```
public class MyServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse
resp)
            throws ServletException, IOException {
        doPost(req, resp);
    }

    @Override
    protected void doPost(HttpServletRequest req,
HttpServletResponse resp)
            throws ServletException, IOException {

        resp.getWriter().write("hello servlet");
    }
```

```
}
```

Web.xml 文件编写

```
<servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>cn.itcast.servlet.MyServlet</servlet-class>

</servlet>

<servlet-mapping>
    <servlet-name>hello</servlet-name>
    <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

访问
http://www.localhost:8080/day04_1/hello

如果是手动进行配置，一定要记得导入 tomcat 下的 lib 下的 servlet-api.jar 文件.

Servlet 生命周期

- init()方法:服务器调用该方法初始化 Servlet
- service()方法:初始化完毕，服务器调用该方法响应客户的请求
- destroy()方法:服务器调用该方法消灭 servlet 对象
- 其中，init()方法只在 Servlet 第一次被请求加载的时候被调用一次，当有客户再请求 Servlet 服务时，Web 服务器将启动一个新的线程，在该线程中，调用 service 方法响应客户的请求
- 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。

示例类代码

```
public class LifeServlet extends HttpServlet {


    public LifeServlet() {

        System.out.println("lifeservlet constructor....");
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse
resp)
            throws ServletException, IOException {
        doPost(req, resp);

    }

    @Override
    protected void doPost(HttpServletRequest req,
HttpServletResponse resp)
            throws ServletException, IOException {
        System.out.println("service...");
    }

    @Override
    public void destroy() {

        System.out.println("destroy...");
    }

    @Override
    public void init() throws ServletException {
        System.out.println("init.....");
    }

}
```

Web.xml 文件配置信息

```
<servlet>
        <servlet-name>life</servlet-name>

    <servlet-class>cn.itcast.servlet.LifeServlet</servlet-class>
        <load-on-startup>2</load-on-startup>
    </servlet>
```

```
<servlet-mapping>
    <servlet-name>life</servlet-name>
    <url-pattern>/life</url-pattern>
</servlet-mapping>
```

On-load-start 介绍

- **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>
```

- **例如：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**

servlet 中 url-pattern 配置

有三种

1.完全匹配　必须使用/开头　/aa/bb

2.目录匹配　必须使用/开头,以*结束　/*　/aa/*

3.扩展名匹配　可以不用/开头，但是必须以*开头　　*.html　*.jsp

最经典错误

/*.do

第一种必须以/开头，如果包含*,它必须放到最后

第二种可以不以/开头，但是它必须是以一个扩展名结束.

完全匹配>目录匹配>扩展名匹配

在后面学习 Filter 会使用的比较多.

web 下的路径问题

在 webRoot 下有一个 admin.html

在 webRoot 下的 a 文件夹下有一个 admin.html

绝对路径

1.http://localhost/day04/admin.jsp ....这种方式一般是用来访问外部

2. 是以/开头，这个/代表的是 tomcat 的根路径.
如果要想访问它下面的 web site,必须的在/后面加上 web site 的名称.

相对路径

1.
http://localhost/day04_1/hello

http://localhost/day04_1/admin.html

上面两个路径是平级的。

要起访问可以写成 ./hello 或 hello

2.

http://localhost/day04_1/hello

http://localhost/day04_1/a/admin.html

要想访问 hello

以上所使用的/ 都是在客户端应用.

我们在 web.xml 中配置 servlet 时也使用/
这个/ 代表的是当前的 web site 根路径 。

Servlet 模板修改

Myeclipse/common/plugins/com.genuitec.eclipse.wizards_9.0.0.me201211011550.jar

1. 使用解压工具打工开
2. 将其中 templates/Servlet.java 复制出来
3. 进行修改
4. 重新将 Servlet.java copy 回去

ServletConfig

常用 API

1.public String getInitParameter(String name) 在 web.xml 中根据 servlet 中配置的 init-param 中配置的名称得到对应的值

2.String getServletName() 获得当前 servlet-name 名称

3.ServletContext getServletContext() 得到 ServletContext 对象

4. Enumeration getInitParameterNames() 得到所有的 init-para 中的 param-name 的值

Servlet 接口中的 init 方法

```
public void init(ServletConfig config) throws ServletException;
```

在 GerericServlet 类中的

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

 public void init(){

 }
```

我在演示 Servlet 生命周期时使用的 init 方法没有参数

int()

原因:我们在 web.xml 文件中配置

```
<servlet>
    <servlet-name>life</servlet-name>
```

```
        <servlet-class>cn.itcast.servlet.LifeServlet</servlet-class>
        <!-- 关于当前 serlvet 的信息 -->
        <init-param>
            <param-name>name</param-name>
            <param-value>tom</param-value>
        </init-param>
    </servlet>
```

tomcat 在启动时会将当前 web site 中的 web.xml 文件加载.
当我们访问一个 servlet 时，tomcat 会根据路径去查找对应的的 Servlet 类
并创建这个类的对象。第一次访问时会调用 init 方法，这时调用的是
有参数的 init 方法,在有参的 init 方法中又调用了无参 init 方法。
所以我们自己写的 init 方法被调用了。

服务器在调用有参 init 方法是传递了一个 ServletConfig 对象，这个对象是
tomcat 自己创建的，我们想要得到这个对象可以直接通过 getServletConfig 方法得到。
得到这个对象后就可以通过 ServletConfig 中提供的 getServletName 方法得到
web.xml 文件中配置的 servlet-name 的值。

可以通过 getInitParamert(String name)得到

```
<init-param>
        <param-name>name</param-name>
        <param-value>tom</param-value>
</init-param>
```

内容中的 value 值。

每一个 Servlet 都有自己的 ServletConfig.通过 ServletConfig 只能得到
自己配置信息.

简单说：就是在自己的 Servlet 中要想得到 ServletConfig 对象 this.getServletConfig()就能得到
得到它以后可以得到 自己在 web.xml 中对当前 Servlet 的相关配置信息.

ServletContext

ServletContext 介绍

常用 API 自己总结
getAttribute
setAttribute
removeAttribute

String getInitParameter(String name)
String getRealPath(String path)

- 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。

- ServletConfig 对象中维护了 ServletContext 对象的引用，开发人员在编写 servlet 时，可以通过 ServletConfig.getServletContext 方法获得 ServletContext 对象。

- 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**。

  ServletContext 是所有的 Servlet 共享的.
  怎样获得一个 ServletContext 可以通过 ServletConfig 对象获得。

  ServletContext 对象存的目的是为了 Servlet 之间进行通信.

练习
统计站点访问次数

关于资源路径

day04_2 工程

1.txt------>day04_2 根目录下
2.txt------>webRoot 下
3.txt------->src 下
4.txt-------->WEB-INF

1.使用 java io 读 eclipse 工程下的这些 txt 文件.

这些文件的路径应该怎样写?

在服务器内通过 getServletContext().getRealPath(“/WEB-INF/info.txt”)
去获得的路径必须是磁盘路径.

1.文件如果在 WebRoot 下，直接写这个文件的名字就可以
2.如果是在 WebRoot 下的子文件夹内 加子文件夹名称

注意：它获得的是服务器所有在的磁盘路径.

类路径 classpath (src 下)
通过字节码对象读取 Class
getResource(“/info.txt”).getFile()

使用 javaIO 读取磁盘文件

```
public class ReadFile {
    public static void main(String[] args) throws Exception {
        // 读取 1.txt
        // readFile("1.txt");
        // 读取 2.txt
        // readFile("WebRoot/2.txt");

        // 读取 3.txt
        // readFile("src/3.txt");

        // 读取 4.txt
        // readFile("WebRoot/WEB-INF/4.txt");

        // 在 eclipse 中它的根目录是 src

        // 使用 classloader 去读文件

    System.out.println(ReadFile.class.getResource("/3.txt").getPat
h()); // 得到 ReadFile 类.class 文件所有的路径下的 3.txt 这个文件.
        readFile(ReadFile.class.getResource("/3.txt").getFile());

    }

    public static void readFile(String filename) throws Exception {

        BufferedReader br = new BufferedReader(new
FileReader(filename));
        String msg = br.readLine();
        System.out.println(msg);
    }
}
```

使用 servlet 读取指定文件

```
public class ReadFileServlet extends HttpServlet {

```

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

        doPost(request, response);
    }

    public void doPost(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        // web 工程的根路径是 WebRoot
        try {
            // 1.txt
            // readFile("");
            // //我们是将 WebRoot 发布到 tomcat 中 1.txt，没有存在于 WebRoot
下.所以在 tomcat 中不可能找到.
            // File file=new File("2.txt");
            // System.out.println(file.getAbsolutePath());

            String filename2 =
this.getServletContext().getRealPath("2.txt");
            System.out.println(filename2);
            // 2.txt
            // readFile(filename2);
            String filename3 = this.getServletContext().getRealPath(
                    "WEB-INF/classes/3.txt");
            System.out.println(filename3);
            // 3.txt
            // readFile(filename3);

            String filename4 = this.getServletContext().getRealPath(
                    "WEB-INF/4.txt");
            // 4.txt
            // readFile(filename4);


    readFile(ReadFile.class.getResource("/3.txt").getFile());

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }
```

```
    public void readFile(String filename) throws Exception {

        BufferedReader br = new BufferedReader(new
FileReader(filename));
        String msg = br.readLine();
        System.out.println(msg);
    }

}
```

第五天 Response 与 Request

Response

- 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

  常用 API
  setStatus(int code) 设置状态码
  addHeader(String key,String value) 一般用来添加多个值
  setHeader(String key,String value) 一般用来设置一个值

示例 1 通过 response 实现请求重定向。

状态码 302
响应头 location=值

示例 2 发送 http 头，控制浏览器定时跳转网页

Refresh:1;url=url 路径

示例 3 发送 http 头，控制浏览器禁止缓存当前文档内容

示例 4 获得向客户端进行数据输出的流对象

- 字节流数据输出
  - OutputStream out = response.getOutputStream();
- 字符流数据输出
  - PrintWriter pw = response.getWriter();

- 指定 body 内容的类型
  - setContentType(“text/html”) // 内容是 html
- 指定输出数据的编码格式
  - setCharacterEncoding("gb2312");
  - 默认情况下，编码格式是 ISO-8859-1

示例 5 输出验证码图片

Request

- 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。

示例 1 获取客户机信息

- getRequestURL 方法返回客户端发出请求完整 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
- getRemoteAddr 方法返回发出请求的客户机的 IP 地址
- getMethod 得到客户机请求方式
- getContextPath 获得工程虚拟目录名称

示例 2 获取请求头信息

- **获得客户机请求头**
  - **getHeader(name)方法 --- String**
  - **getHeaders(String name)方法 --- Enumeration<String>**
  - **getHeaderNames 方法 --- Enumeration<String>**
- **获得具体类型客户机请求头**
  - **getIntHead(name)方法 --- int**
  - **getDateHead(name)方法 --- long(日期对应毫秒)**

示例 3 获取请求参数

- getParameter(name) --- String 通过 name 获得值
- getParameterValues --- String[ ] 通过 name 获得多值 checkbox
- getParameterNames --- Enumeration<String> 获得所有 name
- getParameterMap --- Map<String,String[ ]> key :name value: 多值
- 数据非空校验
- 处理中文乱码
    - post
        - setCharacterEncoding //放在 getParameter 前才有效
    - get
        - new String(str.getBytes("ISO-8859-1"),"utf-8")
        - 设置 tomcat Connector URIEncoding="utf-8"

示例 4 利用请求域传递对象

- 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
- 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，从而共享请求中的数据

请求重定向和请求转发的区别

- 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，它是相对于服务器的根目录；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。

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

- include 在程序执行上效果类似 forward,但是使用 forward 只有一个程序可以生成响应，include 可以由多个程序一同生成响应 ----- 常用来页面布局

第六天　HttpSession 与 Cookie

JSP 与 EL 表达式介绍

Jsp 原理与翻译规则

JSP 运行原理分析
客户端访问服务器端 JSP 文件，服务器会读取 JSP 内容，将其翻译为一个 Servlet，执行 JSP 过程实际上执行 Servlet 过程，返回给客户端页面由 Servlet 程序生成的 。
* 翻译后 Servlet 保存 tomcat/work 目录中
hello.jsp ------ hello_jsp.java
1.jsp ------- _1_jsp.java

Jsp 三种脚本元素

1、<%! %> JSP 声明脚本 翻译成员变量、成员方法、内部类
2、<%= %> JSP 脚本表达式 输出表达式值到 HTML 源代码 ------ 会被翻译为 out.print
* out 是 JSP 内置对象，用于生成 HTML 源代码

3、<% %> JSP 程序脚本代码 ，可以嵌入任意 Java 代码

EL 表达式快速入门

EL 表达式语言是 JSP2.0 新特性 ，主要用来对 JSP 中各种数据范围的对象 进行取值访问 !!!
<%=request.getAttribute(“name”);>
等价于
${requestScope.name}
区别：如果 name 不存在时 ，<%=request.getAttribute(“name”);>会在页面上显示 null，${requestScope.name} 会显示""

Cookie

Cookie 与 Session 技术概述

- Cookie
  - Cookie 是客户端技术，程序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

Cookie 案例 记录上次访问时间

Cookie API 详解

- 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
- ✓ setPath 与 getPath 方法
- ✓ setDomain 与 getDomain 方法
- ✓ getName 方法

- 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。
- 删除持久 cookie，可以将 cookie 最大时效设为 0，注意，删除 cookie 时，path 必须一致，否则不会删除

1 读取 cookie
request.getCookies() 返回 Cookie[]
首先判断 cookies 数组是否存在 cookies == null，如果 cookies 存在，根据 cookie 的 name 去查找指定 cookie

2、服务器向客户端发送 cookie
cookie 对象创建 new Cookie(name,value)
response.addCookie(cookie) 将 cookie 发送客户端

* cookie 有 name 和 value，提供三个方法 getName getValue setValue

3、cookie 从持久性上分为两类 会话 cookie 和持久 cookie
会话 cookie 保存在浏览器内存中 cookie，当会话结束浏览器关闭，会话 cookie 信息就是丢失
持久 cookie 保存在浏览器临时文件缓存区中 cookie （硬盘上），当关闭浏览器结束会话，持久 cookie 不会被删除
* 持久 cookie 存在过期时间，过期后会自动删除
持久 cookie 需要设置 cookie 有效期 setMaxAge

Set-Cookie: lastvisit=1350617047968; Expires=Fri, 19-Oct-2012 04:24:07 GMT

4、cookie 访问有效路径
携带 cookie 必须 path 一致
默认 http://localhost/day6/showLastTime 生成 cookie ---- 默认 path 就是/day6/ (lastTime 资源所在目录就是默认 path)
* 只有在访问 http://localhost/day6/ 目录和子目录情况下 才会携带 cookie 信息

path 也可以手动指定 setPath
new LastVisitCookie.setPath("/abc");

抓取信息：
Set-Cookie: lastvisit=1350617427109; Path=/abc
再次访问 ：http://localhost/day6/visit 不会携带 cookie 信息，因为 path 不符合

最简单方案：setPath("/");
Set-Cookie: lastvisit=1350617644468; Expires=Fri, 19-Oct-2012 04:34:04 GMT; Path=/

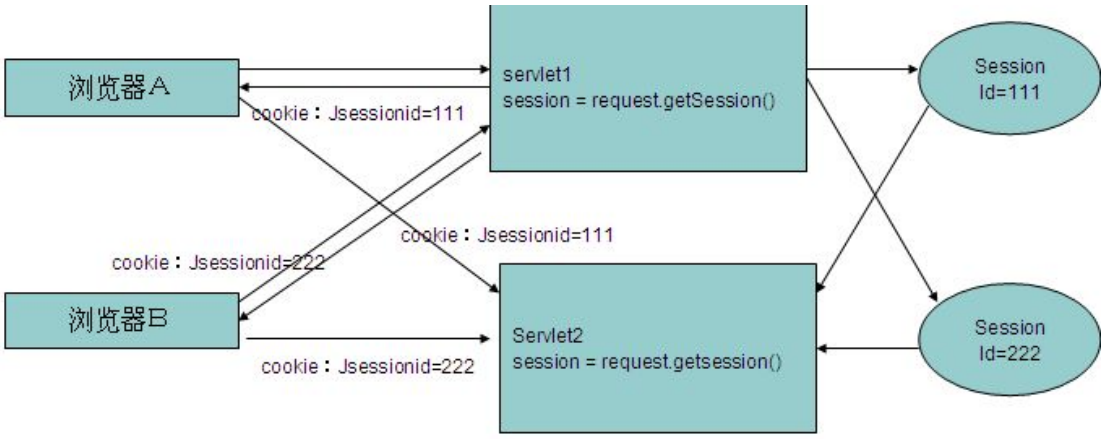
5、cookie 有效域名
setDomain 设置 cookie 有效域名 ---- 例如：setDomain(".itcast.cn"); 传智播客网站的 cookie * 不需要记这个
访问 A 网站 ，生成 B 网站 Cookie ----- 第三方 cookie （恶意 cookie）
访问 A 网站， 生成 A 网站 Cookie ----- 第一方 Cookie（安全 cookie ）

6、删除持久 cookie
删除持久 cookie，可以将 cookie 最大时效设为 0，注意，删除 cookie 时，path 必须一致，否则不会删除

Cookie 案例 商品浏览记录

Session

Session 原理



Session 示例 购物车

Session 追踪

浏览器禁用 Cookie 之后，Session 还能否使用 ？ 可以的

- 浏览器无法保存 cookie 中 jsession id ，无法完成 Session 追踪

通过程序生成 URL （携带 jsessionid 的 URL ） ： URL 重写

使用 URL 重写，必须该网站所有路径都使用 URL 重写 ，实际应用中很少采用

Session 的生命周期
1、Session 对象创建 ： request.getSession() 执行时 （当前会话第一次执行）
2、session 对象何时销毁？
浏览器如果关闭后，Session 对象是不是就销毁了？
答案：不是，Session 保存在服务器端，和浏览器是否关闭没有关系 ，关闭浏览器时删除会话 cookie，丢失 jsessionid，没有 jsession 无法找到服务器端对应 Session

三种销毁 Session 对象情况：1、不正常关闭服务器（正常关闭服务器 Session 信息会被序列化到硬盘中 保存 tomcat/work 目录）
2、Session 过期 默认过期时间在 tomcat/conf/web.xml 配置

```
<session-config>
        <session-timeout>30</session-timeout>
</session-config>
```

* 默认 Session 对象过期时间 30 分钟 （连续不使用 Session 对象时间）
* 也可以手动设置 setMaxInactiveInterval(int interval)

3、在程序中执行 session.invalidate() 手动销毁 Session 对象

Session 示例 Session 实现一次性验证码用户登陆

Servlet 的数据访问范围

Servlet 三种数据访问范围：ServletContext 、HttpSession、HttpServletRequest
1、保存 ServletContext 数据 ，在服务器关闭时才会删除，生命周期最长，全局都可以访问（最少使用）
* 网站访问次数、全局数据库连接池 需要保存 ServletContext
2、保存 HttpSession 数据 ，三种情况下丢失 ，主要保存用户相关数据 (不建议存放大规模数据)
* 用户登录信息、购物信息 保存 HttpSession
3、保存 HttpServletRequest，当前请求发生时产生，响应结束数据立刻释放 (生命周期最短，最建议使用)

* Servlet 获得数据处理结果，通过请求转发 传递信息给 JSP 显示

三种数据范围提供相同几个方法
setAttribute
getAttribute
removeAttribute

第七天 jsp/EL 表达式/EL 函数

JSP

JSP 回顾

- JSP 由来？JSP 和 Servlet 技术区别关系？

Servlet 技术在生成动态网页时，需要通过 response 的输出流，通过 print 语句生成 HTML 源代码
Servlet 生成 HTML 源代码，编程非常复杂，不利于美工页面人员维护 ，也不能使用 Dreamweaver 所见即所得开发工具进行调试
sun 公司为了简化动态网页生成，推出 JSP 技术 ，可以说 JSP 技术就是 Servlet 技术，功能完全相同

- 2、JSP 运行原理

JSP 在执行时，会被服务器翻译为 Servlet 编译执行， JSP 以 Servlet 方式运行的 ------ JSP 就是 Servlet
* JSP 支持 HTML 语法，通过<% %> 嵌入 java 程序代码，Servlet 本身是一个 java 程序，不支持 HTML
* JSP 在运行时 翻译 Servlet 程序 保存 tomcat/work 目录

- 3、JSP 脚本元素

<%! %> JSP 声明 <%= %> JSP 脚本表达式 <% %> 脚本代码
<%! %> 生成内容 翻译 Servlet 程序成员变量、成员方法、内部类
<%= %> 翻译为 out.print ，在 service 方法内部，用于生成 HTML 页面源代码
<% %> 嵌入任何 java 程序代码，运行多个脚本块共同组合为一个 java 程序

JSP 注释

- JSP 注释：<%-- … --%>，转化阶段消失，只能被开发人员看到
- JAVA 注释：//、/**/、/***/，编译阶段消失

- HTML 注释：<!-- … -->, 不会消失，在页面中也能看到

JSP 注释 在 JSP 翻译为 Servlet 代码后 消失 ，只存在于 JSP 源代码中
Java 注释 在 Servlet 程序执行后消失，不会被显示客户端 HTML 源代码中
HTML 注释 会在客户端 HTML 源代码中保留 （在浏览器客户端执行 ）
* HTML 注释 所以不能阻止 JSP 代码和 Java 代码的执行

JSP 指令

- 功能
  - 用于指示 JSP 执行某些步骤
  - 用于指示 JSP 表现特定行为
- 语法格式
  - <%@ directive [attribute =“value”] * %>
- 分类
  - page 指令标记
  - include 指令标记
  - taglib 指令标记

➢ page 指令

- page 属性包含在“<%@ page ”和“%>”之间。
- 这些属性可以单独使用，也可以几个或多个同时使用
- page 指令用来定义 JSP 文件的全局属性
- 在 JSP 页面中，只有 import 可以出现多次，其它属性都只能出现一次

1) language 值必须是 java ，可以不写默认也是 java

2) extends 指定 JSP 翻译后 Servlet 继承哪个类，tomcat 中默认继承 extends org.apache.jasper.runtime.HttpJspBase，一般不做修改，如果非要改，必须继承 HttpServlet

3) session 指定 JSP 翻译后 Servlet 是否可以直接使用内置对象 session，默认 true 可以直接使用
* 如果 session = true 翻译后生成以下两句代码
HttpSession session = null;
session = pageContext.getSession();

4) import 指定 JSP 翻译 Servlet 后 导入包
* 如果使用 当前包和 java.lang 包 之外的类，必须通过 import 导包
import javax.servlet.*;
import javax.servlet.http.*;

import javax.servlet.jsp.*;

* javax.servlet 包 、javax.servlet.http 包、javax.servlet.jsp 包 当中类，在 JSP 中使用时 不需要导包

5) buffer 和 autoFlush 设置 jsp 中内置对象 out 的缓冲区大小 和是否自动输出

* pageContext = _jspxFactory.getPageContext(this, request, response, null, false, 8192, true);

该方法最后两个参数，8192 就是默认 buffer 大小，true 代表 autoFlush 自动输出

<%@ page buffer="16kb" autoFlush="false" %> 生成代码如下：

pageContext = _jspxFactory.getPageContext(this, request, response,null, false, 16384, false);

6) errorPage 和 isErrorPage 设置 JSP 发生错误后跳转页面，设置错误的处理页面（错误友好页面 ）

* 单个页面的友好页面设置，每个 jsp 都需要配置，十分麻烦
* 通用错误页面配置 web.xml

```
<error-page>
        <error-code>500</error-code>
        <location>/500.jsp</location>
</error-page>
```

* 错误处理页面中，通过设置 isErrorPage=true 使用 JSP 内置对象 exception 获得异常信息

```
Throwable exception = org.apache.jasper.runtime.JspRuntimeLibrary.getThrowable(request);
    if (exception != null) {
       response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
    }
```

* IE 浏览器无法显示错误友好页面解决：工具--internet 选项 -- 高级 -- 将显示友好 HTTP 错误信息前面钩 去掉

7) contentType 和 pageEncoding : pageEncoding JSP 和 Servlet 源文件编码，contentType 设置 Http 响应内容编码

8) isELIgnored 设置是否解析 EL 表达式，如果设置 false 会对 EL 进行解析，如果设置 true，不会对 EL 进行解析

➢ include 指令

- include 指令的语法格式如下
  - <%@ include file=“filename” %>
- include 指令的作用是在 JSP 页面中静态包含一个文件，同时由 JSP 解析包含的文件内容
- 静态包含的含义
  - file 不能为一变量

- <% String url="index.html" ; %>
- <%@ include file = "<%= url %>" %>

- 不可以在 file 所指定的文件后接任何参数
  - <%@ include file = "jw.jsp?nm=browser" %>

1) 包含内容注意： 被包含页面不需要 完整 html 结构 <html> <body> ， 只需要编写 HTML 代码片段就可以了

2) include 指令包含原理 （静态包含原理）

* file 属性不能使用变量，不能携带参数

➢ taglib 指令

用来引入标签定义文件

<%@ taglib uri="" prefix="" >

uri : 标签文件名称空间

prefix : 前缀

JSP 内置对象

什么叫内置对象：JSP 翻译为 Servlet 后，在代码中已经存在对象，JSP 页面可以直接使用这些对象

- request HttpServletRequest
- response HttpServletResponse
- session HttpSession
- application ServletContext
- config ServletConfig
- page this (HttpServlet)
- pageContext PageContext
- exception Throwable (所有异常父类)
- out JspWriter

service 方法参数内： request、response

方法内部定义 6 个

```
PageContext pageContext = null;
HttpSession session = null;
ServletContext application = null;
ServletConfig config = null;
JspWriter out = null;
Object page = this;
```

如果 jsp 设置 isErrorPage=true

```
Throwable exception = org.apache.jasper.runtime.JspRuntimeLibrary.getThrowable(request);
if (exception != null) {
   response.setStatus(HttpServletResponse.SC_INTERNAL_SERVER_ERROR);
}
```

Servlet 三种数据范围 ： request、session、ServletContext
JSP 存在四种数据范围 ： page request session application
* application 就是 ServletContext
* page 代表当前页面，保存数据到当前页面数据范围， 该数据只能在当前页面使用

- Page 对象

- **“page”对象代表了正在运行的由 JSP 文件产生的类对象 【一般不建议使用】**
- **page 对象是指向当前 JSP 程序本身的对象 this**
- **page 对象其实是 java.lang.Object 类的实例对象**

- pageContext 对象

pageContext 代表当前页面上下文
1、常用操作一 ：向 JSP 四个范围 存取数据
setAttribute(String name, Object value, int scope)
getAttribute(String name, int scope)

*** setAttribute(String name, Object value) 没有指定 scope，默认将数据保存到 page 范围**

提供了 非常重要方法 findAttribute(String name) ： 按照 page - request - session - application 搜索 指定属性，找到了返回，如果都没有返回 null

2、常用操作二：获取其它八个隐含对象
*** 意义：编写框架，获得 pageContext 一个对象，可以获得 JSP 其它八个内置对象**
- **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**

- out 对象
- **向客户端输出数据**
- **管理服务器输出缓冲区**
- **内部使用 PrintWriter 对象来输出文本级数据**
- **通过 page 指令的 buffer 属性来调整缓冲区的大小，默认的缓冲区是 8kb**

- Exception

- **exception 对象是 java.lang.Trowable 类的实例**
- **(使用前 isErrorPage="true")**
- **exception 对象用来处理 JSP 文件在执行时所有发生的错误和异常**

- **exception 对象可以和 page 指令一起使用，通过指定某一个页面为错误处理页面，对错误进行处理**
    - **<%@ page isErrorPage="true "%>的页面内使用**

JSP 动作标签

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，避免在 JSP 页面中直接编写 java 代码，造成 jsp 页面难以维护。

- JSP 常用标签
- <jsp:useBean>
    - 使用一个 ID 和一个给定作用范围和同一 ID 的 JavaBean 相关联
- <jsp:setProperty>
    - 设置 JavaBean 的属性值
- <jsp:getProperty>
    - 取得 JavaBean 的属性值
- <jsp:include>
    - 请求时文件包含
- <jsp:forward>
    - 接受用户输入并将请求分派给另一页面
- <jsp:param>

Jsp:include

语法 ： <jsp:include page="url" />

jsp:include 动态包含，包含目标 jsp 执行结果

@include 静态包含，包含目标 jsp 翻译 Servlet 源代码

Include 动作与指令区别

- <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:forward

将请求传递给另一个 JSP 页面

jsp:forward 标签取到 JSP 中转发的代码

<jsp:forward page="/jsp/12.jsp"></jsp:forward> 等价于

request.getRequestDispatcher("/jsp/12.jsp").forward(request,response);

* <jsp:forward>之后的代码不执行
* 传递参数通过 request.setAttribute 传递

EL 表达式

- EL 全名为 Expression Language。EL 主要作用：
- 获取数据：
  - EL 表达式主要用于替换 JSP 页面中的**脚本表达式，以从各种类型的 web 域中检索 java 对象、**获取数据。(某个 web 域 中的对象，访问 javabean 的属性、访问 list 集合、访问 map 集合、访问数组)
- 执行运算：
  - 利用 EL 表达式可以在 JSP 页面中执行一些基本的关系运算、逻辑运算和算术运算，以在 JSP 页面中完成一些简单的逻辑运算。${user==null}
- 获取 web 开发常用对象
  - EL 表达式定义了一些隐式对象，利用这些隐式对象，web 开发人员可以很轻松获得对 web 常用对象的引用，从而获得这些对象中的数据。
- 调用 Java 方法
  - EL 表达式允许用户开发自定义 EL 函数，以在 JSP 页面中通过 EL 表达式调用 Java 类的方法。

EL 注意事项

- EL 表达式是 JSP 2.0(JavaEE1.4)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" %>

JavaEE1.4(Servlet2.4/JSP 2.0)
JavaEE5.0(Servlet2.5/JSP 2.1)
JavaEE6.0(Servlet3.0/JSP 2.2)

EL 从 JavaEE1.4 开始 被纳入官方规范，JavaEE1.4 之前，web 页面默认不支持 EL ------使用 EL 必须通过 isELIgnored=false

web 工程中如何识别 web 工程 javaee 版本 ---- web.xml
<web-app version="2.5"
xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"> ========== Servlet2.5

====== JavaEE5.0

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd"> ==== Servlet2.3 ==== JavaEE1.3

获取数据

- 使用 EL 表达式获取数据语法：“${标识符}”
- EL 表达式语句在执行时，会调用 pageContext.findAttribute 方法，用标识符为关键字，分别从 page、request、session、application 四个域中查找相应的对象，找到则返回相应对象，找不到则返回”” （注意，不是 null，而是空字符串）。
- EL 表达式也可以很轻松获取 JavaBean 的属性，或获取数组、Collection、Map 类型集合的数据，例如：
- ${user.address.city}
- ${user.list[0]}：访问**有序集合某个位置**的元素
- ${map.key} ： 获得 map 集合中指定 key 的值
- . 和 [ ] 区别

  //数组和 list 集合 必须用[]取值

  如果属性名称 含有. 特殊字符，必须用[]

- 结合 JSTL 的 foreach 标签，使用 EL 表达式也可以很轻松迭代各种类型的数组或集合，示例：
  - 迭代数组
  - 迭代 collection 类型集合
  - 迭代 map 类型集合
-

执行运算

算术运算： + = * / %
比较运算： > < >= <= == !=
逻辑运算：&& || !

> gt greater than
< lt less than
>= ge greater eaquals
<= le less equals
== eq equals
!= ne not equals
&& and
|| or

！ not

* EL 执行运算时，首先将内容转换为数字，再运算
* ${a + 10} 如果 a 不存在，运算的结果是 10
* 运算必须写在一个 {}中， 典型错误 ${ ${a} + ${b} }

empty 运算符，判断对象是否不存在，集合是否为空
三元表达式 ： ${(empty username)? "用户名不存在":username }

获取 web 开发常用对象

11 个内置对象
pageContext

获得四个数据范围的数据
pageScope
requestScope
sessionScope
applicationScope
* ${pageScope.name } ====== pageContext.getAttribute("name")

==========
问题：getParameter 和 getAttribute 区别 ？
getParameter 获得 HTTP 中请求参数的值 ： get 参数 url?name=zhangsan post 参数位于 HTTP 请求体 name=zhangsan
* form 表单提交、<a href="url?id=1" >

getAttribute 获得服务器端各种数据范围的值 request.getAttribute 获得 request 数据范围的值
* 值在服务器端通过 request.setAttribute 保存的

getParameter 获得客户端提交的数据，getAttribute 服务器内部传递的数据
==========

param 、paramValues 用户获得请求参数的值
${param.name} ==== request.getParameter("name");
${paramValues.name} ===== request.getParameterValues("name");

header、headerValues 获得请求头信息的数据
${header["user-agent"]} ========== request.getHeader("user-agent");
${headerValues["user-agent"]} ========== request.getHeaders("user-agent");

cookie 用来在开发中快速获得 cookie 的值 ------- 是一个 Map<String,Cookie>

${cookie.name.value } 获得指定名称的 cookie 的 value 值

${cookie.name}得到的是一个 cookie 对象.

initParam 用来快速读取 ServletContext 的全局初始化参数
${initParam.name } ==========
getServletContext().getInitParameter("name");
* <context-param> 配置全局参数

最常使用：cookie ---------- ${cookie.name.value } 这里 name 是 cookie 的 name 值
pageContext ---------- ${pageContext.request.contextPath } ====
pageContext.getRequest().getContextPath() 返回 /day8
<!-- 编写一个链接 -->
link
<!-- 使用 EL 获得工程名 -->
link

快速获得 ip ： ${pageContext.request.remoteAddr }

调用 java 方法

- **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。**
- **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**

编写自定义 EL 函数
1、编写静态 java 方法

2、在 tld 文件中描述 java 方法
*** tld 文件放到 WEB-INF 下**
新建 tld 时 shortname 访问函数前缀 URI 名称空间
将新建 tld taglib 头进行修改
<taglib version="2.0" 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">

******* 参考 tomcat/webapps/examples/WEB-INF/jsp2 有一个文件**
<function>
<!-- 自定义 EL 函数名称 -->
<name>show</name>
<!-- EL 函数对应 java 类完整类名 -->
<function-class>cn.itcast.utils.ShowUser</function-class>
<!-- static 静态方法签名 -->
<function-signature>java.lang.String
show(java.lang.String)</function-signature>
</function>

3、在 jsp 中引用 tld 文件，调用 EL 函数
<!-- 通过 taglib 引用 tld 文件 -->
<%@taglib uri="http://www.itcast.cn/myel" prefix="myel" %>
调用 自定义 show 格式 : ${prefix:方法名(参数)}

在 sun 提供 JSTL 标签库中自带了一套 EL 函数库 （十六个和字符串操作相关函数）
1、在 web 工程中导入 jstl jar 包
将 jstl.jar 和 standard.jar 复制 WEB-INF/lib 下

2、在 JSP 中引用 EL 函数库标签描述文件 (tld 文件)
<!-- 引用 EL 官方 tld 文件 -->
%@taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %

第八天 JSTL/自定义标签/软件国际化

JSTL 标签库

JSTL 标签库 + EL 学习目的， 规范 JSP 的编写，在 JSP 内部 不写 <%%> Java 代码

- JavaServer Pages Standard Tag Library
- 由 JCP（Java Community Process）指定标准
- 提供给 Java Web 开发人员一个标准通用的标签函数库
- 和 EL 来取代传统直接在页面上嵌入 Java 程序（Scripting）的做法，以提高程序可读性、维护性和方便性
- 下载
    - JSTL 主要由 Apache 组织的 Jakarta Project 实现

    - http://tomcat.apache.org/taglibs/standard/
    - 容器必须支持 Servlet 2.4 且 JSP 2.0 以上版本
    - JavaEE1.4
- 安装
    - 解压缩后将 lib 中的 jstl.jar、standard.jar 复制到 WEB 应用程序的 WEB-INF\lib 下
- **核心标签库** (core) --- c
- **国际化标签 fmt**
- **数据库标签 sql --Servlet**
- XML 标签 xml
- JSTL 函数(EL 函数) el

因为数据库操作和 XML 操作已经都用 Servlet 来实现了，不用 JSP，所以不会使用 sql 和 xml 标签库
使用 JSP 主要用来进行页面显示 只学习 core 和 fmt
今天主要学习 core
会在后面国际化课程中介绍 fmt 标签库

JSTL core 标签库中 12 个标签
1、c:out 输出内容到浏览器端
2、c:set 设置值到四种数据范围，设置四个范围中 java 对象的一个属性值
3、c:remove 用来删除四个数据范围的数据
4、c:catch 捕获程序异常
5、c:if 取代页面中 if 条件语句，记住没有 c:else
<c:if test="${m>5}" > </c:if>
6、c:choose c:when c:otherwise 一起使用 ，取代 if elseif else 结构 、switch-case 结构
7、c:forEach 取代页面中 for 循环
8、c:forTokens 切割字符串
9、c:import 效果和 include 类似
10、c:url 完成 URL 重写（禁用 cookie 的 session 追踪 ） ---- 结合 c:param 完成 中文 URL 编码
11、c:redirect 完成请求重定向

自定义标签

简单标签（实现 SimpleTag）

- 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----------→传入 pageContext
- setParent 和 getParent 方法----------→将父标签引入
- setJspBody 方法----------→将标签体内容缓存到内存对象中
- doTag 方法----------→标签操作
-

在 setJspBody 方法中传入对象 JspFragment ，该对象封装了 标签体内容，控制标签体内容输出

***** 最重要方法 invoke(Writer out) 意义将标签体内容输出到指定字符输出流中

注意：在 简单标签库中 <bodyContent> 不能写 JSP，而必须写 scriptless

➢ 简单标签体系

SimpleTag

|----------SimpleTagSupport

使用简单标签开发和传统标签一样的四个常见功能

➢ 示例一：控制页面部分代码是否执行 ----- jspFragment.invoke(传入 out 对象);

什么都不写，页面上不会执行标签体

如果希望执行：

getJspBody().invoke(getJspContext().getOut()); // 将内容输出到页面

简化为

getJspBody().invoke(null); 如果输出流为 null，采用默认输出流 out 对象

➢ 示例二：控制标签后 JSP 页面是否继续执行

什么都不写就会继续执行

如果不想继续执行 throw new SkipPageException();

➢ 示例三：重复执行标签体

```
for (int i = 0; i < 5; i++) {
    getJspBody().invoke(null);
}
```

➢ 示例四：修改标签体内容输出 转换为大写

```
// 获得缓存中标签体内容
StringWriter stringWriter = new StringWriter();
getJspBody().invoke(stringWriter);// 将标签体内容写入 stringwriter
String content = stringWriter.toString();

// 转换为大写
content = content.toUpperCase();

// 输出到页面 需要 out 对象
this.getJspContext().getOut().print(content);
```

标签库应用练习

- 示例 1 开发 html 转义标签

对一段 HTML 代码进行转义，将转义后内容输出到页面 在页面上显示 HTML 代码
参考 tomcat\webapps\examples\WEB-INF\classes\util\HTMLFilter.java

示例 2. 截取字符串

做一个标签

<my:substring start=”1” end=”4” msg=”abcdef”/>------→结果 bcd

打包标签库

将标签库制作为一个 jar 包，像 JSTL 一样，在以后项目中复用该标签库
1、新建 java 工程 tagproject
2、在 java 工程中 新建 META-INF 目录，将编写标签 tld 文件 复制到该目录
3、将标签类 复制 java 工程 src 目录 （需要为 java 工程导入 javaee 的 jar 包 ）
右键工程属性 --- Java Build Path --- liberaries --- add Liberary 添加 myeclipse liberaries (添加 javaee5 类库)
4、右键工程 export 选项 导出 jar file

国际化(I18N)

- 软件的国际化：软件开发时，要使它能同时应对世界不同地区和国家的访问，并针对不同地区和国家的访问，提供相应的、符合来访者阅读习惯的页面或数据。
- 国际化又称为 i18n：internationalization
- 软件实现国际化，需具备哪些特征：
  - 对于程序中固定使用的文本元素，例如菜单栏、导航条等中使用的文本元素、或错误提示信息，状态信息等，需要根据来访者的地区和国家，选择不同语言的文本为之服务。
  - 对于程序动态产生的数据，例如(日期，货币等)，软件应能根据当前所在的国家或地区的文化习惯进行显示。

将软件中使用文本信息，保存 properties 属性文件中（key=value）
为不同国家编写不同 properties 属性文件，许多国家 properties 文件 组成一个资源包

ResourceBundle 会根据来访者国家不同，而自动读取不同 properties 文件

资源包中所有 properties 文件必须具有相同基名 basename, 在 basename 后可以通过_拼接国家和语言信息 例如：basename_语言_国家.properties
myproperties_zh.properties 中文的配置文件 这里 zh 就是语言
myproperties_en.properties 英文的配置文件 这里 en 就是语言
myproperties_zh_CN.properties 中国中文配置文件 这里 zh 语言 CN 国家

myproperties.properties 没有国家和语言信息，默认资源文件，如果来访者的国家和语言的资源文件没有找到，就会读取默认资源文件

- 资源文件书写格式

- 资源文件的内容通常采用“关键字＝值”的形式，软件根据关键字检索值显示在页面上。一个资源包中的所有资源文件的关键字必须相同，值则为相应国家的文字。
- 并且资源文件中采用的是 properties 格式文件，所以文件中的所有字符都必须是 ASCII 字码，对于像中文这样的非 ACSII 字符，须先进行编码。(java 提供了一个 native2ascII 命令用于编码)。属性文件是不能保存中文的

固定文本的国际化

1、加载 src 下 properties 文件
ResourceBundle bundle = ResourceBundle.getBundle(basename);
2、读取 properties 文件中内容
String value = bundle.getString(key);
* 也可以在 ResourceBundle.getBundle 时 传入一个代表国家和地区 Locale 对象
* ResourceBundle bundle = ResourceBundle.getBundle("info", Locale.CHINA); 指定读取中国配置文件

优先级：指定 Locale > 系统默认区域和语言 > 资源包默认的

- 示例 登录页面国际化

使用 fmt 国际化标签 实现国际化登陆页面
1、将 jstl 的 jar 复制 WEB-INF/lib
2、引用 fmt 标签库 <%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
设置 locale
<fmt:setLocale value="${param.locale}"/>
创建 ResourceBundle
<fmt:setBundle basename="info" var="bundle" scope="page"/>
显示 message
<fmt:message bundle="${bundle}" key="login.info" />
<fmt:message bundle="${bundle}" key="login.username" />

日期格式化

- DateFormat 类可以将一个日期/时间对象格式化为表示某个国家地区的日期/时间字符串。

- DateFormat 类除了可按国家地区格式化输出日期外，它还定义了一些用于描述日期/时间的**显示模式**的 int 型的常量，包括 FULL, LONG, MEDIUM, DEFAULT,

SHORT，实例化 DateFormat 对象时，可以使用这些常量，控制日期/时间的显示长度

- 实例化 ***DateFormat*** 类有九种方式，以下三种为带参形式，下面列出的三种方式也可以分别不带参，或只带显示样式的参数。
  - getDateInstance(int style, Locale aLocale)：以指定的日期显示模式和本地信息来获得 DateFormat 实例对象，该实例对象不处理时间值部分。
  - getTimeInstance(int style, Locale aLocale)：以指定的时间显示模式和本地信息来获得 DateFormat 实例对象，该实例对象不处理日期值部分。
  - getDateTimeInstance(int dateStyle, int timeStyle, Locale aLocale)：以单独指定的日期显示模式、时间显示模式和本地信息来获得 DateFormat 实例对象。

数字格式化

- 实例化 NumberFormat 类时，可以使用 locale 对象作为参数，也可以不使用，下面列出的是使用参数的。
- getNumberInstance(Locale locale)：以参数 locale 对象所标识的本地信息来获得具有多种用途的 NumberFormat 实例对象
- getIntegerInstance(Locale locale)：以参数 locale 对象所标识的本地信息来获得处理**整数**的 NumberFormat 实例对象
- getCurrencyInstance(Locale locale)：以参数 locale 对象所标识的本地信息来获得处理**货币**的 NumberFormat 实例对象
- getPercentInstance(Locale locale)：以参数 locale 对象所标识的本地信息来获得处理**百分比数值**的 NumberFormat 实例对象

练习

- 请创建一个 date 对象，并把 date 对象中表示日期部分的时间值，以及表示时间部分的时间值，分别以 short、long 模式进行格式化输出（国家设置为中国）。
- 请将时间值：09-11-28 上午 10 时 25 分 39 秒 CST，反向解析成一个 date 对象。
- 请将整数 198，输出为货币形式：$198，并将$198 反向解析成整数 198。
- 请将 0.78654321，输出百分比格式，保留两位小数

动态文本国际化

允许信息中 将动态数据 抽取出来，使用{0} {1} {2} 占位符进行占位，达到动态生成效果

1) MessageFormat.format(pattern,args); 采用默认 Locale 进行国际化

2) MessageFormat formater = new MessageFormat(pattern,locale); // 执行国家
formater.format(args); 采用指定 Locale 进行国际化

占位符国际化复杂写法：{0,类型,样式}

类型为 number 样式 integer、currency、percent
类型为 date 样式 full long medium short
类型为 timer 样式 full long medium short

第九天 java web 模式与示例

JSP 两种模式介绍

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

JavaBean 介绍

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。

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。

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。

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。

userBean

- <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。

setProperty

- <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。

getProperty

<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。

内省技术与 BeanUtil 使用

内省

- 访问 JavaBean 属性的两种方式：
  - 直接调用 bean 的 setXXX 或 getXXX 方法。
  - 通过内省技术访问(java.beans 包提供了内省的 API)
    - 内省技术是基于反射技术的
    - 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。

编程步骤：

1、通过 Introspector 获得 JavaBean 对象 BeanInfo

```
BeanInfo beanInfo = Introspector.getBeanInfo(Person.class);
```

2、通过 BeanInfo 获得 JavaBean 对象 所有属性描述器 PropertyDescriptor

```
PropertyDescriptor[] propertyDescriptors = beanInfo.getPropertyDescriptors();
```

3、遍历每个属性描述器，可以获得属性名称 getName()

4、通过属性描述器 获得属性读写方法

```
Method setter = propertyDescriptor.getWriteMethod();
Method getter = propertyDescriptor.getReadMethod();
```

BeanUtil

- 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。
- Beanutils 工具包的常用类：
    - BeanUtils
        - populate(Object bean, Map properties)
    - 自定义转换器
        - ConvertUtils.regsiter(Converter convert, Class clazz)

模式二与 MVC 思想

- Model2 分为三部分
    - Servlet 控制显示哪个 JSP 页面给用户、调用哪个 Javabean
    - JSP 响应用户请求，把结果数据展现给用户
    - JavaBean 对数据库的数据的存取，复杂的业务功能和逻辑处理

model2 程序开发步骤

1、客户端提交请求给 Servlet
2、Servlet 接收数据，封装数据到 JavaBean 中
3、Servlet 调用 JavaBean 处理数据，得到处理结果
4、Servlet 将数据处理结果传递给 JSP
5、JSP 生成响应页面，返回客户端

- Model-View-Controller 的简称
- MVC 是一种设计模式
- 把应用程序分成三个核心模块
    - 模型、视图、控制器

编码实战

用户管理（jsp+servlet+javabean+el+jstl+dom4j+xpath）

用户名唯一
用户信息
Username 用户名
Password 密码
Age 年龄
Hobby 爱好

Birthday 生日
Email 邮箱

用户注册功能

1. 验证
2. 验证码

用户登录功能

登录成功，显示登录用户信息，提供注销功能。

自动登录(可选)

用户删除与修改(可选)

第十天 数据库入门 MySql

数据库介绍

什么是数据库？ 就是一个文件系统，通过标准 SQL 语言操作文件系统中数据 ---- 用来存放软件系统的数据
什么是关系化数据库 ？ 保存关系数据模型

Oracle 甲骨文公司，专门数据库厂商 收购 BEA 、SUN、MySQL ------- 收费 大型数据库 ，用于任何系统任何平台
MySQL 早期开源免费数据库产品，LAMP 组合 Linux + Apache + MySQL + PHP 完全开源免费 ，自从 mysql 被 oracle 收购后，从 6.0 开始出现收费版本
DB2 IBM 数据库产品 大型收费数据库 websphere 服务器一起使用
SYBASE 中等规模数据库 收费 PowerDesigner 数据库建模工具
SQL Server 微软公司数据库产品 收费中等规模数据库 ，操作系统要求是 windows 结合.net 一起使用

Java 开发者主要使用 MySQL 、Oracle、DB2 三种数据库

Mysql 安装与配置

- 卸载

在 mysql 安装目录 my.ini

datadir="C:/Documents and Settings/All Users/Application Data/MySQL/MySQL Server 5.5/Data/"
1) 在控制面板中卸载 MySQL
2) 删除 mysql 安装目录
3) 删除 MySQL 数据文件目录

➢ 2、安装
选择自定义安装
MySQL Server 默认位置 c:\program files\mysql 目录
Server Data File 数据文件 C:\Documents and Settings\All Users\Application Data\MySQL 目录

➢ 安装后进行 mysql 配置
1) 配置 mysql 默认字符集
默认 latin1 等价于 ISO-8859-1 改为 utf8
2) Include Bin Directory in Window Path 将 mysql/bin 目录配置环境变量 path ---- 勾选
3) 输入超级管理 root 密码

测试 mysql 是否安装成功 启动 cmd 窗口 输入 mysql -u root -p 回车 输入密码 root ==== 出现 mysql> 安装成功

➢ 3、重置 root 密码
1) 停止 mysql 服务器 运行输入 services.msc 停止 mysql 服务
2) 在 cmd 下 输入 mysqld --skip-grant-tables 启动服务器 光标不动（不要关闭该窗口）
3) 新打开 cmd 输入 mysql -u root -p 不需要密码
use mysql;
update user set password=password('abc') WHERE User='root';
4) 关闭两个 cmd 窗口 在任务管理器结束 mysqld 进程
5) 在服务管理页面 重启 mysql 服务
密码修改完成

SQL 语言

Sql 分类

- DDL （数据定义问题）
  - 数据定义语言 - Data Definition Language
  - 用来定义数据库的对象，如数据表、视图、索引等
- DML （数据操纵问题）
  - 数据处理语言 - Data Manipulation Language
  - 在数据库表中更新，增加和删除记录
  - 如 update， insert， delete
- DCL （数据控制问题）

    - 数据控制语言 – Data Control Language
    - 指用于设置用户权限和控制事务语句
    - 如 grant，revoke，if…else，while，begin transaction
- DQL （数据查询问题）
    - 数据查询语言 – Data Query Language
    - Select

数据库操作语句

1、创建数据库 会为每个软件系统创建单独数据库
语法 ： create database 数据库名称 ； (创建数据库采用数据库服务器默认字符集)
复杂写法 create database 数据库名称 character set 字符集 collate 比较规则 ;

创建一个名称为 mydb1 的数据库。 create database mydb1;
创建一个使用 utf8 字符集的 mydb2 数据库。 create database mydb2 character set utf8;
创建一个使用 utf8 字符集，并带校对规则的 mydb3 数据库。create database mydb3 character set utf8 collate utf8_bin;

补充：每次创建一个数据库在 数据存放目录中生成一个文件夹 ，每个文件夹中存在 db.opt 存放默认字符集和校对规则
datadir="C:/Documents and Settings/All Users/Application Data/MySQL/MySQL Server 5.5/Data/"

2、查询数据库
show databases; ----- 查看所有数据库
show create database 数据库名; ------ 查看数据编码集

3、删除数据库
语法 ：drop database 数据库名称;

查看当前数据库服务器中的所有数据库 show databases;
查看前面创建的 mydb2 数据库的定义信息 show create database mydb2;
删除前面创建的 mydb1 数据库 drop database mydb1;

4、修改数据库编码集
语法：alter database 数据库名称 character set 字符集 collate 比较规则;

修改 mydb2 字符集为 gbk; alter database mydb2 character set gbk;

切换当前使用数据库： use 数据库名称
查看当前正在使用数据库: select database();

数据表结构 SQL 语句

1、数据表的创建
语法：create table 表名(列名 类型(长度),列名 类型(长度)...);
一个数据表 可以存在很多列，每列具有类型和长度
* 创建表时没有指定 字符集，将采用数据库默认字符集
* 创建表之前 必须使用 use db 语法指定操作数据库

创建 day12 数据库 create database day12;
切换到 day12 数据库 use day12;

例如：
User {
    id   int
    name string
    password string
    birthday date
}
一个 java 类 对应数据库中一张数据表，一个 java 对象 对应数据表中一条数据记录

MySQL 常用数据类型
java 中 String char ----- mysql 中字符串型 char varchar
* char 是定长 varchar 是变长
例如：char(8) 保存 lisi，因为 lisi 只有四个字符，所有会补充四个空格，成为 8 个字符存入 char(8)中 ，如果有 varchar(8) 自动根据存放内容改变长度

java 中 byte short int long float double ----- mysql 中数值类型 TINYINT 、SMALLINT、INT、BIGINT、FLOAT、DOUBLE

java 中 boolean ---- mysql 逻辑性 bit 存放一位数值 0 或者 1

java 中 Date ----- mysql 日期类型 date (只有日期) time(只有时间) datetime(日期时间都有) timestamp(日期时间都有)
* datetime 和 timestamp 表现形式上完全相同，区别就在于 timestamp 在数据库可以自定更新（当前时间）

java 中 大数据类型 inputStream 二进制文件 Reader 文本文件 ------- mysql 大数据类型 blob(存放大二进制数据) text(存放大的文本文件)
* tinyblob tinytext 255 字节 blob text 64KB mediumblob mediumtext 16MB longblob longtext 4GB

例如：
User {
    id   int   ------ mysql int

```
    name string    ------ mysql varchar
    password string    ----- mysql varchar
    birthday date        ----- mysql date
}
```

员工表建表语句

```
create table employee (
    id int,
    name varchar(20),
    gender varchar(20),
    birthday date,
    entry_date date,
    job varchar(30),
    salary double,
    resume longtext
);
```

查看表结构 desc 表名;

*** 创建数据表时，只有字符串类型必须写长度，而其他类型都有默认长度

2、单表创建时约束

约束用来保证数据有效性和完整性

主键约束 primary key : 信息记录某个字段可以唯一区分其他信息记录，这个字段就可以是主键 （唯一 非空）

唯一约束 unique : 该字段的值不允许重复

* 一张表中可以有很多个唯一约束，只能有一个(两个)作为主键约束

非空约束 not null ： 该字段的值不能为空

```
create table employee2 (
    id int primary key auto_increment,
    name varchar(20) unique not null,
    gender varchar(20) not null,
    birthday date not null,
    entry_date date not null,
    job varchar(30) not null,
    salary double not null,
    resume longtext
);
```

* 如果主键约束类型为 数值型 int bigint ，添加 auto_increment 自动增长

3、数据表结构修改

1) 增加列 语法: alter table 表名 add 列名 类型(长度) 约束;

2) 修改现有列类型、长度和约束 语法：alter table 表名 modify 列名 类型(长度) 约束;
3) 修改现有列名称 语法：alter table 表名 change 旧列名 新列名 类型(长度) 约束;
4) 删除现有列 语法：alter table 表名 drop 列名 ;
5) 修改表名 rename table 旧表名 to 新表名;

* 修改表的字符集：alter table student character set utf8;

在上面员工表的基本上增加一个 image 列。---- alter table employee add image varchar(100) ;
修改 job 列，使其长度为 60。 ----alter table employee modify job varchar(60) not null;
删除 gender 列。 ----alter table employee drop gender ;
表名改为 user。 ----rename table employee to user;
修改表的字符集为 utf8 ---- alter table user character set utf8;
列名 name 修改为 username ----- alter table user change name username varchar(20) unique not null;

查看当前数据库内所有表：show tables
查看当前数据表的详细信息 ： show create table user;

4、数据表删除
语法：drop table 表名;

5、查看数据表结构
desc 表名; 查看表结构
show tables ; 查看当前库内所有表名
show create table 表名; 查看建表语句和字符集

数据表中数据记录的增删改查操作

1、向数据表插入记录
语法：insert into 表名(列名,列名,列名...) values(值,值,值...); 为数据表的每列进行赋值

注意事项
1) 插入值 类型必须和 列类型匹配
2) 值长度不能超过 列定义长度
3) 值的顺序和 列顺序对应
4) 字符串和日期型值 必须写 单引号
5) 插入空值 可以写 null

新建 employee 表，插入三个员工信息
insert into employee(id,name,gender,birthday,entry_date,job,salary,resume) values(1,'zhangsan','male','1990-10-10','2010-01-01','sales',4000,'good boy !');

语法 2: 省略所有列名，但是后面值必须要和表中所有列进行匹配，按照表中列顺序
insert into employee values(2,'lisi','male','1988-10-01','2008-08-17','hr',3500,'good hr !');

语法 3：省略可以为空，有默认值部分列名，后面值要和前面列进行匹配
insert into employee(id,name,job,salary) values(3,'wangwu','boss',20000);

* 在插入记录后 ，通过 select * from employee; 查看所有员工信息

插入一条中文记录
insert into employee(id,name,job,salary) values(4,'小明','清洁员',1500);
出错了：
ERROR 1366 (HY000): Incorrect string value: '\xC3\xF7' for column 'name' at row 1 ;
错误原因：mysql client 采用默认字符集编码 gbk

查看系统所有字符集 ： show variables like 'character%';

解决办法：修改客户端字符集为 gbk
MYSQL 中共有 6 个地方字符集 ：client connetion result 和客户端相关 、database server system 和服务器端相关

第一种：当前窗口临时修改 set names gbk ;
* 只对当前窗口有效，关闭后就会失效

第二种：配置 mysql/my.ini 文件
[mysql] 客户端配置
[mysqld] 服务器端配置

修改客户端字符集 [mysql] 后字符集 default-character-set=gbk

使用 mysql -u root -p 密码连入数据库后，如果进行数据库操作，直接操作，如果要进行数据表结构和数据记录操作，必须先切换到操作的数据库 use db;

2、数据记录更改操作
语法: update 表名 set 列名=值,列名=值.... where 条件语句;
* 如果没有 where 条件语句，默认修改所有行数据

将所有员工薪水修改为 5000 元。 ----- update employee set salary = 5000;
将姓名为’zhangsan’的员工薪水修改为 3000 元。 ------- update employee set salary = 3000 where name='zhangsan' ;
将姓名为’lisi’的员工薪水修改为 4000 元,job 改为 ccc。------- update employee set salary=4000, job='ccc' where name='lisi';
将 wangwu 的薪水在原有基础上增加 1000 元。 ---------- update employee set salary = salary+1000 where name ='wangwu';

3、数据记录的删除操作
语法：delete from 表名 where 条件语句 ;

* 如果没有 where 语句，将删除表中 所有记录

如果要删除表中所有数据记录，使用 truncate table 表名; 等价于 delete from 表名;

试题：如果使用 delete 删除表中所有记录和使用 truncate table 删除表中所有记录 有何不同 ？
truncate 删除数据，过程先将整个表删除，再重新创建
delete 删除数据，逐行删除记录
* truncate 效率要好于 delete
truncate 属于 DDL ，delete 属于 DML ======== 事务管理只能对 DML 有效 ，被事务管理 SQL 语句可以回滚到 SQL 执行前状态

删除表中名称为’zhangsan’的记录。 ------ delete from employee where name='zhangsan';
删除表中所有记录。----- delete from employee; (可以事务回滚)
使用 truncate 删除表中记录。 ---- truncate table employee;

数据表记录的查询

语法一 ： select [distinct] * | 列名,列名... from 表名;
select * from 表名; 查询该表中所有列信息
select 列名,列名... from 表名; 查询表中指定列的信息
distinct 用于排重

```
create table exam(
    id int primary key auto_increment,
    name varchar(20) not null,
    chinese double,
    math double,
    english double
);
```

```
insert into exam values(null,'关羽',85,76,70);
insert into exam values(null,'张飞',70,75,70);
insert into exam values(null,'赵云',90,65,95);
```

查询表中所有学生的信息。 --------- select * from exam;
查询表中所有学生的姓名和对应的英语成绩。 ----- select name,english from exam;
过滤表中重复数据 (查询英语成绩，排除完全相同重复数据) ---- select distinct english from exam;

语法二：select 表达式(列名执行运算) from 表名;
select 列名 as 别名 from 表名;

在所有学生分数上加 10 分特长分。 ---- select name,chinese+10,math+10,english+10 from exam;
统计每个学生的总分。------- select name,chinese+math+english from exam;
使用别名表示学生分数。----- select name,chinese+math+english as 总分 from exam;

*** 在对列起别名时，as 可以省略 select name,chinese+math+english as 总分 from exam;
------ select name,chinese+math+english 总分 from exam;
select name,math from exam; 查询 name 和 math 两列的值
select name math from exam; 查询 name 列值，起别名 math

语法三: select 列名 from 表名 where 条件语句

查询姓名为关羽的学生成绩 -------select * from exam where name='关羽';
查询英语成绩大于 90 分的同学 ----- select * from exam where english > 90;
查询总分大于 200 分的所有同学 ----- select * from exam where chinese+math+english > 200;

运算符
1) 相等= 不等 <>
2) between ...and... 在两者之间取值 between 70 and 80 等价于 >=70 <=80 ----- 注意前面那个数要比后面那个数要小
3) in(值,值,值) 在指定值中任取一个 in(70,80,90) 值可以是 70、80 或者 90
4) like '模糊查询 pattern' 进行模糊查询 ，表达式有两个占位符 % 任意字符串 _ 任意单个字符 例如： name like '张%' 所有姓张学员
name like '张_' 所有姓张名字为两个字学员
5) is null 判断该列值为空

6) and 逻辑与 or 逻辑或 not 逻辑非

查询英语分数在 90－100 之间的同学。 -------- select * from exam where english>=90 and english <= 100; select * from exam where english between 90 and 100;
查询数学分数为 65,75,85 的同学。 ---- select * from exam where math in(65,75,85);
查询所有姓赵的学生成绩。---- select * from exam where name like '赵%';
查询英语分>80，语文分>80 的同学。 ---- select * from exam where english > 80 and chinese > 80;

insert into exam values(null,'刘备',null,55,38);
查询语文没有成绩学员 select * from exam where chinese is null;
查询语文有成绩学员 select * from exam where chinese is not null;

语法四： select * from 表名 order by 列名 asc|desc ; ---- asc 升序 desc 降序
对数学成绩排序后输出。---------- select * from exam order by math; 默认 asc 升序
对总分排序按从高到低（降序）的顺序输出 ---------- select * from exam order by math+chinese+english desc;

对学生成绩按照英语进行降序排序，英语相同学员按照数学降序 ---------- select * from exam order by english desc,math desc;

聚集函数 指 SQL 语句中内置函数 ---------- 分组函数（用于统计）
1) count 统计查询结果记录条数 select count(*)|count(列名) from 表名;

统计一个班级共有多少学生？ ---------- select count(*) from exam;
统计英语成绩大于 90 的学生有多少个？ ------- select count(*) from exam where english > 90;
统计总分大于 220 的人数有多少？ --------select count(*) from exam where chinese+math+english > 220;

2) sum 统计某一列数据的和 select sum(列名) from 表名;

统计一个班级数学总成绩？ ----- select sum(math) from exam;
统计一个班级语文、英语、数学各科的总成绩 ---- select sum(chinese),sum(math),sum(english) from exam;
统计一个班级语文、英语、数学的成绩总和 select sum(chinese+math+english) from exam;
select sum(chinese)+sum(math)+sum(english) from exam;
**** 刘备语文 null ，null 进行所有运算 结果都是 null
select sum(chinese)+sum(math)+sum(english) from exam; 含有刘备英语和数学成绩
select sum(chinese+math+english) from exam; 不含刘备英语和数学成绩
* 使用 ifnull 函数处理 null 情况
select sum(ifnull(chinese,0)+ifnull(math,0)+ifnull(english,0)) from exam; 含有刘备英语和数学成绩

统计一个班级语文成绩平均分 ------ select sum(chinese)/count(*) from exam;

3) avg 统计某一列平均值 select avg(列名) from 表名;
求一个班级数学平均分？ ---- select avg(math) from exam;
求一个班级总分平均分？---- select avg(ifnull(chinese,0)+ifnull(math,0)+ifnull(english,0)) from exam;

4) max 统计一列最大值 min 统计一列最小值
求班级最高分和最低分（数值范围在统计中特别有用） select max(chinese+math+english) ,min(ifnull(chinese,0)+ifnull(math,0)+ifnull(english,0)) from exam;

语法五：select 分组函数 from exam group by 列名; 按照某列进行分组统计
分组操作，就是具有相同数据记录分到一组中，便于统计

```
create table orders(
    id int,
    product varchar(20),
    price float
);
```

insert into orders(id,product,price) values(1,'电视',900);
insert into orders(id,product,price) values(2,'洗衣机',100);
insert into orders(id,product,price) values(3,'洗衣粉',90);
insert into orders(id,product,price) values(4,'桔子',9);
insert into orders(id,product,price) values(5,'洗衣粉',90);

练习：对订单表中商品归类后，显示每一类商品的总价 ---- 需要按照商品名称进行分组
select product,sum(price) from orders group by product;

在 group by 语句后面 添加 having 条件语句 ---- 对分组查询结果进行过滤
练习：查询购买了几类商品，并且每类总价大于 100 的商品
select product,sum(price) from orders group by product having sum(price) > 100;

试题：where 和 having 条件语句的区别 ？
where 是在分组前进行条件过滤，having 是在分组后进行条件过滤
使用 where 地方都可以用 having 替换 ， 但是 having 可以使用分组函数，而 where 后不可以用分组函数

小结 select 语句 ： S-F-W-G-H-O 组合 select ... from ... where ... group by... having... order by ... ;
顺序不能改变
解析顺序 ： from - where - group by - having –select- order by

MySQL 数据库的备份和恢复

1、备份命令 mysql/bin/mysqldump 将数据库 SQL 语句导出
语法：mysqldump -u 用户名 -p 数据库名 > 磁盘 SQL 文件路径

例如： 备份 day12 数据库 --- c:\day12.sql
cmd > mysqldump -u root -p day12 > c:\day12.sql 回车输入密码

2、恢复命令 mysql/bin/mysql 将 sql 文件导入到数据库
语法： mysql -u 用户名 -p 数据库名 < 磁盘 SQL 文件路径
***** 导入 SQL 必须手动创建数据库 ，SQL 不会创建数据库

例如：将 c:\day12.sql 导入 day12 数据库
cmd > mysql -u root -p day12 < c:\day12.sql 回车密码

补充知识：恢复 SQL 也可以在数据库内部执行 source c:\day12.sql

多表设计（外键约束）

```
create table emp (
   id int primary key auto_increment,
   name varchar(20),
   job varchar(20),
   salary double,
     dept_id int,
     foreign key emp(dept_id) references dept(id)
);

insert into emp values(null,'小丽','人力资源',4500,1);
insert into emp values(null,'小张','Java 工程师',5000,3);
insert into emp values(null,'老李','财务经理',8000,2);
insert into emp values(null,'小刘','项目经理',10000,3);

create table dept(
     id int primary key auto_increment,
     name varchar(20)
);

insert into dept values(null,'人力资源部');
insert into dept values(null,'财务部');
insert into dept values(null,'技术研发部');
```

让员工表和部门表发生关系，知道员工属于哪个部门 ，在员工表添加部门 id 字段

```
alter table emp add dept_id int ;
update emp set dept_id = 1 where name = '小丽';
update emp set dept_id = 2 where name = '老李';
update emp set dept_id = 3 where name = '小刘';
update emp set dept_id = 3 where name = '小张';
```

公司因为财政问题，解散技术研发部 delete from dept where name ='技术研发部'; ----- 小张和小刘 失去了组织

emp 表 中 dept_id 字段 引用 dept 表 id 字段 ------- 添加外键约束 （保证数据有效和完整性）

将 emp 表中 dept_id 设置为外键约束 alter table emp add foreign key(dept_id) references dept(id) ;

无法删除技术研发部 ，因为小刘和小张信息 依赖技术研发部 记录 !!!!!

多表设计原则 ：所有关系数据只能存在三种对应关系 （一对一、一对多、多对多）
1、多对多关系 ： 雇员和项目关系
一个雇员可以参与多个项目
一个项目可以由多个雇员参与

建表原则：必须创建第三张关系表，在关系表中引用两个实体主键 作为外键
关系表中每条记录，代表一个雇员参与了一个项目

2、一对多关系： 用户和博客关系
一个用户可以发表多篇博客
一个博客只能由一个作者

建表原则：不需要创建第三方关系表，只需要在多方添加 一方主键作为 外键

3、一对一关系 ： 这种关系很少见到 负责人和工作室
一个负责人 管理一个工作室
一个工作室 只有一个负责人

建表规则：在任一方添加对方主键 作为外键

多表查询

多表查询——笛卡尔积
将 A 表中每条记录 与 B 表中每条记录进行 匹配 获得笛卡尔积
select * from emp;
select * from dept;

select * from emp,dept; 显示结果就是笛卡尔积
笛卡尔积结果 就是 两个表记录乘积 例如 A 表 3 条 B 表 4 条 ---- 笛卡尔积 12 条

笛卡尔积结果是无效的，必须从笛卡尔积中选取有效的数据结果 !!!

多表查询 连接查询 内连接查询
从 A 表中选择一条记录，去 B 表中找对应记录 ----- 内连接 必须 A 表和 B 表存在对应记录才会显示

```
create table A(A_ID int primary key auto_increment,A_NAME varchar(20) not null);
insert into A values(1,'Apple');
insert into A values(2,'Orange');
insert into A values(3,'Peach');

create table B(A_ID int primary key auto_increment,B_PRICE double);
insert into B values(1,2.30);
```

insert into B values(2,3.50);
insert into B values(4,null);

使用内连接 select * from a,b where a.a_id = b.a_id;
* 内连接查询结果条数 一定小于 两个表记录较少哪个表 ----- 例如 A 表 3 条 B 表 5 条 ---- 内连接结果条数 <= 3

select * from emp,dept where emp.dept_id = dept.id ; 将 emp 表和 dept 表进行内连接
在内连接查询时 添加条件，查询人力资源部有哪些 员工 ？？ select * from emp,dept where emp.dept_id = dept.id and dept.name ='人力资源部';
工资大于 7000 员工来自哪个部门？ select * from emp,dept where emp.dept_id = dept.id and emp.salary > 7000;

select * from emp,dept where emp.dept_id = dept.id ; 写法一
select * from emp inner join dept on emp.dept_id = dept.id ; 写法二

第十一天 JDBC 介绍

JDBC 介绍

- JDBC 全称为：Java DataBase Connectivity（java 数据库连接）。
- SUN 公司为**了简化、**统一对数据库的操作，定义了一套 Java 操作数据库的规范，称之为 JDBC。

学习 JDBC 技术目的，使用 Java 技术操作数据库中数据记录

什么是驱动？ 两个设备要进行通信，满足一定通信数据格式，数据格式由设备提供商规定，设备提供商为设备提供驱动软件，通过软件可以与该设备进行通信

如果没有 JDBC，Java 程序员需要面向各个数据库驱动接口编程，开发复杂 ； sun 公司提供一套统一 JDBC 接口规范，Java 程序只需要使用 JDBC 就可以操作任何数据库，JDBC 实现类由各个数据库厂商提供，

学习 JDBC
1、学习 JDK 中自带 JDBC 接口规范 java.sql javax.sql
DriverManager 驱动管理类
Connection 连接接口
Statement (PreparedStatement、CallableStatement) 数据库操作
ResultSet 结果集接口

2、必须在工程中引入不同数据库驱动实现

JDBC 体验

- **编程从 user 表中读取数据，并打印在命令行窗口中。**

```
create table user(
    id int primary key auto_increment,
    username varchar(20) unique not null,
    password varchar(20) not null,
    email varchar(40) not null
);
```

一、搭建实验环境 ：

1、在 mysql 中创建一个库，并创建 user 表和插入表的数据。

2、新建一个 Java 工程，并导入数据驱动。

二、编写程序，在程序中加载数据库驱动

DriverManager. registerDriver(Driver driver)

三、建立连接(Connection)

Connection conn = DriverManager.getConnection(url,user,pass);

四、创建用于向数据库发送 SQL 的 Statement 对象，并发送 sql

Statement st = conn.createStatement();

ResultSet rs = st.executeQuery(sql);

五、从代表结果集的 ResultSet 中取出数据，打印到命令行窗口

六、断开与数据库的连接，并释放相关资源

JDBC API 详解(重点)

DriverManager 类

static void registerDriver(Driver driver)　注册一个 JDBC 驱动程序

注意：DriverManager 中可以同时注册多个 JDBC 驱动 例如：同时注册 mysql、oralce、db2 驱动 ，通过对 JDBC URL 分析，决定采用哪个驱动

static Connection getConnection(String url, String user, String password)　根据 jdbc url 和 用户名、密码获得一个数据库连接

实际开发中，不推荐使用 DriverManager.registerDriver 会导致驱动注册两次、会使得程序依赖 具体数据库 API

推荐使用 ：Class.forName("com.mysql.jdbc.Driver"); 加载 Driver 类时完成驱动注册，使得程序不依赖 MySQL 的 API

***** 不要引入 与数据库相关 具体 API

JDBC URL

jdbc:mysql://localhost:3306/day13
这里 jdbc: 是 JDBC 连接协议
这里 mysql:// 是 mysql 数据库连接协议，JDBC 子协议
localhost:3306 主机和端口
day13 数据库

常用数据库 URL 写法
MYSQL jdbc:mysql://localhost:3306/day13
ORACLE jdbc:oracle:thin:@localhost:1521:sid

创建数据表 users

```
create table users(
    id int primary key ,
    username varchar(20) unique not null,
    password varchar(20) not null,
    email varchar(40) not null
);
```

插入一些数据记录

```
insert into users values(1,'zhangsan','123','zhangsan@itcast.cn');
insert into users values(2,'lisi','123','lisi@itcast.cn');
insert into users values(3,'wangwu','123','wangwu@itcast.cn');
```

oracle 中执行 insert update delete 后 必须使用 commit 操作

4) 在工程引入 oracle 的 jdbc 驱动
安装目录\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar

5) 修改 Oracle 驱动类 oracle.jdbc.driver.OracleDriver 和 URL jdbc:oracle:thin:@localhost:1521:orcl

MySQL 如果连接 localhost:3306 可以省略
jdbc:mysql://localhost:3306/day13 ---------- jdbc:mysql:///day11
JDBCURL 可以通过?和& 携带参数
常用属性：useUnicode=true&characterEncoding=UTF-8 ---------- 解决操作数据库乱码问题

Connection 连接接口

应用一：获得 SQL 的操作对象
Statement conn.createStatement() 该对象可以将 SQL 发送给数据库进行执行
PreparedStatement conn.prepareStatement(sql) 对 SQL 语句进行预编译，防止 SQL 注入
CallableStatement conn.prepareCall(sql); 该对象可以调用数据库中存储过程 （以后 Oracle 学习）

应用二：对数据库事务进行管理（明天）
conn.setAutoCommit(boolean); 设置事务是否自动提交
conn.commit(); 提交数据库事务
conn.rollback(); 回滚数据库事务

Statement 用于将 SQL 发送给数据库 获得操作结果

发送单条 SQL
executeUpdate 用于向数据库发送 insert update delete 语句，返回 int 类型参数，代表影响记录行数
executeQuery 用于向数据库发送 select 语句，返回 ResultSet 结果集对象
execute 用于数据库发送任何 SQL 语句（包括 DDL DML DCL） 返回 boolean ，SQL 执行结果是 ResultSet 返回 true，否则 false

发送多条 SQL
addBatch(sql) 将 SQL 加入批处理队列
executeBatch() 执行队列中所有 SQL 语句 ，一次性向数据库发送多条 SQL

使用 ResultSet 遍历结果集

```
while(rs.next()){
    // 根据数据库内部 列类型，选择相应 getXXX 方法
    int ---- getInt
    varchart ---- getString
    date ----- getDate
}
```

在 java.sql 定义 Date、Time 、TimeStamp 对应数据库中 date time timestamp 类型 ---------- java.sql.Date/Time/TimeStamp 都是 java.util.Date 子类
java.sql.Date 只有日期没有时间
java.sql.Time 只有时间没有日期
java.sql.TimeStamp 既有日期也有时间

getXXX 有两种写法 第一种 getString(index) 结果集中列索引 第二种 getString(列名)

思考：如果 SQL 语句可能会返回一行数据，也可能查不到任何记录时，代码应该怎么写？

```
----- 用于登陆
if(rs.next()){
  // 查到了数据
}else{
  // 没有查到数据
}
```

ResultSet 高级应用 ---- 滚动结果集

Connection 接口的 createStatement() 返回 Statement 对象，操作 SQL 后 产生 ResultSet 默认执行 next 向前滚动，不支持在滚动中对数据进行修改 （只读不执行滚动）
Connection 接口还提供 createStatement(int resultSetType, int resultSetConcurrency) 在创建 Statement 对象 设置结果集类型，并发策略

结果集类型
ResultSet.TYPE_FORWARD_ONLY 只能向前，只能调用 next 不能向回滚动
ResultSet.TYPE_SCROLL_INSENSITIVE 支持结果集向回滚动，不能查看修改结果
ResultSet.TYPE_SCROLL_SENSITIVE 支持结果集向回滚动，查看修改结果

结果集并发策略
ResultSet.CONCUR_READ_ONLY 只读
ResultSet.CONCUR_UPDATABLE 支持修改

常见三种组合
ResultSet.TYPE_FORWARD_ONLY 和 ResultSet.CONCUR_READ_ONLY （默认） 只读不支持向回滚动
ResultSet.TYPE_SCROLL_INSENSITIVE 和 ResultSet.CONCUR_READ_ONLY 只读，支持向回滚动
ResultSet.TYPE_SCROLL_SENSITIVE 和 ResultSet.CONCUR_UPDATABLE 支持向回滚动，支持对数据修改

JDBC 完成 CRUD 示例

编写对 user 表 增删改查程序，从重复代码中提取公共方法 JDBCUtils 工具类 ，将数据库连接参数写入 properties 配置文件

示例代码

DAO 模式

DAO 模式（Data Access Object 数据访问对象）：在持久层通过 DAO 将数据源操作完全封装起来，业务层通过操作 Java 对象，完成对数据源操作
* 业务层无需知道数据源底层实现 ，通过 java 对象操作数据源

DAO 模式结构 ：
1、数据源（MySQL 数据库）
2、Business Object 业务层代码，调用 DAO 完成 对数据源操作
3、DataAccessObject 数据访问对象，持久层 DAO 程序，封装对数据源增删改查，提供方法参数都是 Java 对象
4、TransferObject 传输对象（值对象） 业务层通过向数据层传递 TO 对象，完成对数据源的增删改查

DAO 登录示例

使用三层结构和 DAO 模式登陆
cn.itcast.web ---- 表现层
cn.itcast.service ---- 业务层
cn.itcast.dao ----- 持久层
cn.itcast.domain ----- 对应数据表实体类 TO 对象

1、编写 login.jsp 登陆表单，提交 /day11/login
2、编写 LoginServlet 获得请求数据，调用业务层 UserService
3、UserService 编写业务逻辑，调用数据层 UserDAO 返回结果
4、在 LoginServlet 获得结果，判断结果跳转页面

示例代码

SQL 注入

由于没有对用户输入进行充分检查，而 SQL 又是拼接而成，在用户输入参数时，在参数中添加一些 SQL 关键字，达到改变 SQL 运行结果的目的，也可以完成恶意攻击。

String sql = select * from user where username =" and password =" ;

例如：
一、输入 username: 老李' or '1'='1 password 随意
select * from user where username ='老李' or '1'='1' and password ='';

* and 优先级 执行 高于 or

二、在 SQL 添加 -- 是 mysql 的注释 输入 username: 老李' -- password 随意
select * from user where username ='老李' -- ' and password ='' ;

解决 SQL 注入：使用 PreparedStatement 取代 Statement
PreparedStatement 解决 SQL 注入原理，运行在 SQL 中参数以?占位符的方式表示
select * from user where username = ? and password = ? ;
将带有?的 SQL 发送给数据库完成编译 （不能执行的 SQL 带有?的 SQL 进行编译 叫做预编译），在 SQL 编译后发现缺少两个参数
PreparedStatement 可以将? 代替参数 发送给数据库服务器，因为 SQL 已经编译过，参数中特殊字符不会当做特殊字符编译，无法达到 SQL 注入的目的

问题：
SQL 注入原理是什么?
1.在输入时连接一个永远为真的一个值
2.使用 mysql 中的 – *注释*
为什么 PreparedStatement 可以防止 SQL 注入 ?
因为它对 sql 语句进行预编译。

JDBC 处理大数据

- 在实际开发中，程序需要把大文本 Text 或二进制数据 Blob 保存到数据库。
- Text 是 mysql 叫法，Oracle 中叫 Clob

- 基本概念：大数据也称之为 LOB(Large Objects)，LOB 又分为：
  - clob 和 blob
    - clob 用于存储大文本。Text
    - blob 用于存储二进制数据，例如图像、声音、二进制文等。

- 对 MySQL 而言只有 blob，而没有 clob，mysql 存储大文本采用的是 Text，Text 和 blob 分别又分为：
    - TINYTEXT(255) 、 TEXT(64k) 、 MEDIUMTEXT(16M) 和 LONGTEXT(4G)
    - TINYBLOB、BLOB、MEDIUMBLOB 和 LONGBLOB

JDBC 进行批处理

- 业务场景：当需要向数据库发送一批 SQL 语句执行时，应避免向数据库一条条的发送执行，而应采用 JDBC 的批处理机制，以提升执行效率。
- 实现批处理有两种方式，第一种方式：

- Statement.addBatch(sql)
- 执行批处理 SQL 语句
- executeBatch()方法：执行批处理命令
- clearBatch()方法：清除批处理命令

- 采用 Statement.addBatch(sql)方式实现批处理：
  - 优点：可以向数据库发送多条不同的ＳＱＬ语句。
  - 缺点：
    - SQL 语句没有预编译。
    - 当向数据库发送多条语句相同，但仅参数不同的 SQL 语句时，需重复写上很多条 SQL 语句。例如：

```
Insert into user(name,password) values(‘aa’,’111’);
Insert into user(name,password) values(‘bb’,’222’);
Insert into user(name,password) values(‘cc’,’333’);
Insert into user(name,password) values(‘dd’,’444’);
```

- 实现批处理的第二种方式：
  - PreparedStatement.addBatch()

- 采用 PreparedStatement.addBatch()实现批处理
  - 优点：发送的是预编译后的 SQL 语句，执行效率高。
  - 缺点：只能应用在 SQL 语句相同，但参数不同的批处理中。因此此种形式的批处理经常用于在同一个表中批量插入数据，或批量更新表的数据。

第十二天 事务与连接池

事务

事务介绍

- 事务的概念
  - 事务指逻辑上的一组操作，组成这组操作的各个单元，要不全部成功，要不全部不成功
- 数据库开启事务命令
  - start transaction　开启事务
  - Rollback　回滚事务
  - Commit　　提交事务

Mysql 中使用事务

1.创建表

```
create table account(
    id int primary key auto_increment,
    name varchar(20),
    money double
);

insert into account values(null,'aaa',1000);
insert into account values(null,'bbb',1000);
insert into account values(null,'ccc',1000);
```

2、MySQL 中事务默认自动提交的，每当执行一条 SQL，就会提交一个事务 （一条 SQL 就是一个事务）

Oracle 中事务默认 不自动提交，需要在执行 SQL 语句后 通过 commint 手动提交事务

3、mysql 管理事务

方式一 ：同时事务管理 SQL 语句

start transaction 开启事务

rollback 回滚事务 （将数据恢复到事务开始时状态）

commit 提交事务 （对事务中进行操作，进行确认操作，事务在提交后，数据就不可恢复）

方式二：数据库中存在一个自动提交变量 ，通过 show variables like '%commit%'; ---- autocommint 值是 on，说明开启自动提交

关闭自动提交 set autocommit = off / set autocommit = 0

如果设置 autocommit 为 off，意味着以后每条 SQL 都会处于一个事务中，相当于每条 SQL 执行前 都执行 start transaction

补充：Oracle 中 autocommit 默认就是 off

Jdbc 使用事务

- 当 Jdbc 程序向数据库获得一个 Connection 对象时，默认情况下这个 Connection 对象会自动向数据库提交在它上面发送的 SQL 语句。若想关闭这种默认提交方式，让多条 SQL 在一个事务中执行，可使用下列语句：
- JDBC 控制事务语句
    - Connection.setAutoCommit(false); // 相当于 start transaction
    - Connection.rollback(); rollback
    - Connection.commit(); commit
- 示例 ***演示银行转帐案例***

- 在 JDBC 代码中使如下转帐操作在同一事务中执行。
  update from account set money=money-100 where name=‘a’;
  update from account set money=money+100 where name=‘b’;
- 设置事务回滚点
  - Savepoint sp = conn.setSavepoint();
  - Conn.rollback(sp);
  - Conn.commit(); //回滚后必须要提交

事务特性

- **原子性（Atomicity）**
  原子性是指事务是一个不可分割的工作单位，事务中的操作要么都发生，要么都不发生。
- **一致性（Consistency）**
  事务前后数据的完整性必须保持一致。
- **隔离性（Isolation）**
  事务的隔离性是指多个用户并发访问数据库时，一个用户的事务不能被其它用户的事务所干扰，多个并发事务之间数据要相互隔离。
- **持久性（Durability）**
  持久性是指一个事务一旦被提交，它对数据库中数据的改变就是永久性的，接下来即使数据库发生故障也不应该对其有任何影响。

事务隔离级别

- 多个线程开启各自事务操作数据库中数据时，数据库系统要负责隔离操作，以保证各个线程在获取数据时的准确性。
- 如果不考虑隔离性，可能会引发如下问题

1、脏读 ：指一个事务读取另一个事务未提交的数据

A 转账 给 B 100，未提交
B 查询账户多了 100
A 回滚
B 查询账户那 100 不见了

2、不可重复读：在一个事务先后两次读取发生数据不一致情况，第二次读取到另一个事务已经提交数据 （强调数据更新 update）

A 查询账户 5000
B 向 A 账户转入 5000
A 查询账户 10000

3、虚读(幻读) ： 在一个事务中，第二次读取发生数据记录数的不同 ，读取到另一个事务已经提交数据 （强调数据记录变化 insert ）

A 第一次读取 存在 5 条记录
B 向 A 插入一条新的记录

A 第二次读取 存在 6 条记录

数据库内部定义了四种隔离级别，用于解决三种隔离问题

1 Serializable：可避免脏读、不可重复读、虚读情况的发生。（串行化）
2 Repeatable read：可避免脏读、不可重复读情况的发生。（可重复读）不可以避免虚读
3 Read committed：可避免脏读情况发生（读已提交）
4 Read uncommitted：最低级别，以上情况均无法保证。(读未提交)

操作数据库内部隔离级别
set session transaction isolation level 设置事务隔离级别
select @@tx_isolation 查询当前事务隔离级别

实验一：演示脏读发生
在 A 窗口 将隔离级别设置 read uncommitted
A、B 窗口同时开启事务
B 窗口执行转账操作 update account set money = money - 500 where name='bbb';
update account set money = money + 500 where name ='aaa'; 未提交事务
A 窗口查询 select * from account; 查询到转账结果（脏读）
B 回滚 rollback
A 窗口查询 金钱丢失

实验二：演示不可重复读
在 A 窗口设置 隔离级别 read committed; 重复实验一，发现无法脏读
B 窗口转账后，提交事务
A 窗口查询到 B 提交的数据 （不可重复读）

实验三：演示阻止不可重复读发生
在 A 窗口 设置隔离级别 repeatable read 重复试验二，发现不会发生不可重复读

实验四：演示 serializable 串行事务效果
A 窗口设置隔离级别 serializable
A、B 同时开启事务
B 窗口插入一条数据 insert into account values(null,'ddd',1000);
在 A 窗口查询数据 select * from account;
发现 A 窗口阻塞了，等待 B 事务执行结束

安全性：serializable > repeatable read > read committed > read uncommitted
性能 ：serializable < repeatable read < read committed < read uncommitted

结论： 实际开发中，通常不会选择 serializable 和 read uncommitted ， mysql 默认隔离级别 repeatable read ，oracle 默认隔离级别 read committed

JDBC 程序中能否指定事务的隔离级别 ？
Connection 接口中定义事务隔离级别四个常量：

static int TRANSACTION_READ_COMMITTED
指示不可以发生脏读的常量；不可重复读和虚读可以发生。
static int TRANSACTION_READ_UNCOMMITTED
指示可以发生脏读 (dirty read)、不可重复读和虚读 (phantom read) 的常量。
static int TRANSACTION_REPEATABLE_READ
指示不可以发生脏读和不可重复读的常量；虚读可以发生。
static int TRANSACTION_SERIALIZABLE
指示不可以发生脏读、不可重复读和虚读的常量。

通过 void setTransactionIsolation(int level) 设置数据库隔离级别

事务的丢失更新问题(lost update)

- 两个或多个事务更新同一行，但这些事务彼此之间都不知道其它事务进行的修改，因此第二个更改覆盖了第一个修改

- 丢失更新问题的解决
- 悲观锁（ Pessimistic Locking ）
*select * from table lock in share mode（读锁、共享锁）*
*select * from table for update （写锁、排它锁）*
- 乐观锁（ Optimistic Locking ）
通过时间戳字段

解决方案详解：
方案一：悲观锁 （假设丢失更新一定会发生 ） ----- 利用数据库内部锁机制，管理事务
方案二：乐观锁 （假设丢失更新不会发生）------- 采用程序中添加版本字段解决丢失更新问题

一、悲观锁
mysql 数据库内部提供两种常用 锁机制：共享锁(读锁)和排它锁(写锁)
允许一张数据表中数据记录，添加多个共享锁，添加共享锁记录，对于其他事务可读不可写的
一张数据表中数据记录，只能添加一个排它锁，在添加排它锁的数据 不能再添加其他共享锁和排它锁的 ，
对于其他事物可读不可写的

*** 所有数据记录修改操作，自动为数据添加排它锁
添加共享锁方式：select * from account lock in share mode ;
添加排它锁方式：select * from account for update;

* 锁必须在事务中添加 ，如果事务结束了 锁就释放了

解决丢失更新：事务在修改记录过程中，锁定记录，别的事务无法并发修改

二、乐观锁
采用记录的版本字段，来判断记录是否修改过 ---------- timestamp
timestamp 可以自动更新

```
create table product (
    id int,
    name varchar(20),
    updatetime timestamp
);

insert into product values(1,'冰箱',null);
update product set name='洗衣机' where id = 1;
```

* timestamp 在插入和修改时 都会自动更新为当前时间

解决丢失更新：在数据表添加版本字段，每次修改过记录后，版本字段都会更新，如果读取是版本字段，与修改时版本字段不一致，说明别人进行修改过数据 （重改）

事务案例：银行转账(在 service 层处理事务 ThreadLocal)

数据库连接池

连接池介绍

一次性创建多个连接，将多个连接缓存在内存中 ，形成数据库连接池（内存数据库连接集合），如果应用程序需要操作数据库，只需要从连接池中获取一个连接，使用后，并不需要关闭连接，只需要将连接放回到连接池中。
* 好处：节省创建连接与释放连接 性能消耗 ---- 连接池中连接起到复用的作用 ，提供程序性能

自定义连接池

- 编写连接池需实现 javax.sql.DataSource 接口。DataSource 接口中定义了两个重载的 getConnection 方法：
    - Connection **getConnection**()
    - Connection **getConnection**(String username, String password)

- 实现 DataSource 接口，并实现连接池功能的步骤：
    - 在 DataSource 构造函数中批量创建与数据库的连接，并把创建的连接保存到一个集合对象中

- 实现 getConnection 方法，让 getConnection 方法每次调用时，从集合对象中取一个 Connection 返回给用户。
- 当用户使用完 Connection，调用 Connection.close()方法时，Connection 对象应保证将自己返回到连接池的集合对象中,而不要把 conn 还给数据库。

自定义连接池程序问题：
1、尽量不要使用具体对象类型的引用
MyDataSource dataSource = new MyDataSource(); 应该写为 DataSource dataSource = new MyDataSource();
2、使用自定义方法 close 将连接放回连接池 ，需要用户在使用时需要记忆额外 API

解决：让用户定义连接池时 DataSource dataSource = new MyDataSource(); 在用户使用连接后，应该调用 conn.close(); 完成将连接放回到连接池
对 close 方法进行方法增强

练习：增强 close 方法，不真正关闭连接，而是将连接放回到连接池
增强 java 中一个方法存在三种方式：1、继承 方法覆盖 2、包装 3、动态代理
1、继承方法覆盖，对原有类方法进行加强
注意：必须控制对象构造，才能使用继承的方式对方法进行加强

2、包装 （装饰者）
编写包装类 1) 必须要与被装饰对象，有相同父类或者实现相同接口 2) 必须将被装饰对象传递给装饰者 （构造函数）
包装是一种通用方法加强措施，不管是否控制目标对象创建，都可以完成方法加强

3、动态代理
原理：利用类加载器 ，在内存中根据目标类 加载器和接口 创建一个代理对象 ，通过代理对象完成对原有对象方法进行加强
注意：被代理对象，必须实现接口
应用场景：也是只有存在目标对象，就可以通过动态代理进行加强

DBCP 连接池

DBCP 是 Apache 的 commons 项目的一个子项目
去官网下载 dbcp 和 pool 的 jar 包 ，DBCP 依赖 POOL 的 jar 包

复制两个 jar 到 WEB-INF/lib 下

DBCP 连接池 核心类 BasicDataSource

* 任何数据库连接池，需要数据库连接，必须通过 JDBC 四个基本参数构造

1、手动设置参数

```
// 使用连接池
BasicDataSource basicDataSource = new BasicDataSource();
// 设置 JDBC 四个基本参数
basicDataSource.setDriverClassName("com.mysql.jdbc.Driver");
basicDataSource.setUrl("jdbc:mysql:///day14");
basicDataSource.setUsername("root");
basicDataSource.setPassword("abc");
```

2、通过配置文件

```
// 根据属性参数 获得连接池
InputStream in = DBCPTest.class.getResourceAsStream("/dbcp.properties");
Properties properties = new Properties();
// 装载输入流
properties.load(in);

DataSource dataSource = BasicDataSourceFactory.createDataSource(properties)
```

C3P 0 连接池

主流开源连接池，在 Hibernate 和 Spring 都提供对 C3P0 连接池支持
去下载 c3p0 开发包 http://sourceforge.net

将 c3p0 的 jar 复制 WEB-INF/lib 下

1、手动

```
// 核心连接池类
ComboPooledDataSource comboPooledDataSource = new ComboPooledDataSource();
// 设置四个 JDBC 基本连接属性
comboPooledDataSource.setDriverClass("com.mysql.jdbc.Driver");
comboPooledDataSource.setJdbcUrl("jdbc:mysql:///day14");
comboPooledDataSource.setUser("root");
comboPooledDataSource.setPassword("abc");
```

2、在 src 下新建 c3p0-config.xml
ComboPooledDataSource comboPooledDataSource = new ComboPooledDataSource(); 会自定加载配置文件

常用基本连接池属性

acquireIncrement	如果连接池中连接都被使用了，一次性增长 3 个新的连接
initialPoolSize	连接池中初始化连接数量 默认:3
maxPoolSize	最大连接池中连接数量 默认：15 连接
maxIdleTime	如果连接长时间没有时间，将被回收 默认：0 连接永不过期
minPoolSize	连接池中最小连接数量 默认：3

Tomcat 内置连接池

tomcat 服务器内置连接池 （使用 Apache DBCP）
配置 tomcat 内置连接池，通过 JNDI 方式 去访问 tomcat 的内置连接池

JNDI Java 命名和目录接口，是 JavaEE 一项技术 ，允许将一个 Java 对象绑定到一个 JNDI 容器（tomcat）中 ，并且为对象指定一个名称
通过 javax.naming 包 Context 对 JNDI 容器中绑定的对象进行查找，通过指定名称找到绑定 Java 对象

操作步骤
1、配置使用 tomcat 内置连接池 配置<context> 元素
context 元素有三种常见配置位置
1) tomcat/conf/context.xml 所有虚拟主机，所有工程都可以访问该连接池
2) tomcat/conf/Catalina/localhost/context.xml 当前虚拟主机(localhost)下所有工程都可以使用该连接池
3) 当前工程/META-INF/context.xml 只有当前工程可以访问该连接池

```
<Context>
   <Resource name="jdbc/EmployeeDB" auth="Container"
                type="javax.sql.DataSource" username="root" password="abc"
                driverClassName="com.mysql.jdbc.Driver" url="jdbc:mysql:///day14"
                maxActive="8" maxIdle="4"/>
</Context>
```

* 必须先将 mysql 驱动 jar 包 复制 tomcat/lib 下
* 在 tomcat 启动服务器时，创建连接池对象，绑定 jdbc/EmployeeDB 指定名称上

2、通过运行在 JNDI 容器内部的程序（Servlet/JSP）去访问 tomcat 内置连接池
Context context = new InitialContext();
Context envCtx = (Context)context.lookup("java:comp/env"); 固定路径
DataSource datasource = (DataSource) envCtx.lookup("jdbc/EmployeeDB"); 通过绑定名称，查找指定 java 对象

第十三天: jdbc/DBUtil 使用/ jdbc 案例

元数据

元数据 MetaData : 指数据库中 库、表、列的定义信息

1、DataBaseMetaData 数据库元数据

1、通过 DataBaseMetaData 获得 数据库连接的基本参数

- getURL()：返回一个 String 类对象，代表数据库的 URL。
- getUserName()：返回连接当前数据库管理系统的用户名。
- getDriverName()：返回驱动驱动程序的名称。
- getPrimaryKeys(String catalog, String schema, String table)：返回指定表主键的 结果集

2、获得数据库、表、列、主键、外键 定义信息

getTables
getColumns
getPrimaryKeys

2、ParameterMetaData 参数元数据

- PreparedStatement . getParameterMetaData()
  - 获得代表 PreparedStatement 元数据的 ParameterMetaData 对象。
  - Select * from user where name=? And password=?
- ParameterMetaData 对象
  - getParameterCount()
    - 获得指定参数的个数
  - getParameterTypeName(int param)
    - 获得指定参数的 sql 类型

3、ResultSetMetaData 结果集元数据(重点..)

- ResultSet. getMetaData()
  - 获得代表 ResultSet 对象元数据的 ResultSetMetaData 对象。
- ResultSetMetaData 对象
  - getColumnCount()
    - 返回 resultset 对象的列数
  - getColumnName(int column)
    - 获得指定列的名称
  - getColumnTypeName(int column)
    - 获得指定列的类型

Dbutil 框架

Dbutil 介绍

- commons-dbutils 是 Apache 组织提供的一个开源 JDBC 工具类库，它是对 JDBC 的简单封装，学习成本极低，并且使用 dbutils 能极大简化 jdbc 编码的工作量，同时也不会影响程序的性能。因此 dbutils 成为很多不喜欢 hibernate 的公司的首选。
- API 介绍：
  - org.apache.commons.dbutils.QueryRunner --- 核心
  - org.apache.commons.dbutils.ResultSetHandler

- 工具类
  - org.apache.commons.dbutils.DbUtils、。

DBUtils 学习
1、QueryRunner 框架核心类 ，所有数据库操作都是必须通过 QueryRunner 进行的
2、ResultSetHandler 结果集封装接口，完成将 ResultSet 结果集 封装为一个 Java 对象
3、DbUtils 工具类 提供驱动管理、事务管理、释放资源等一系列公共方法

DbUtils

- DbUtils ：提供如关闭连接、装载 JDBC 驱动程序等常规工作的工具类，里面的所有方法都是静态的。主要方法如下：
  - public static void close(…) throws java.sql.SQLException： DbUtils 类提供了三个重载的关闭方法。这些方法检查所提供的参数是不是 NULL，如果不是的话，它们就关闭 Connection、Statement 和 ResultSet。
  - public static void closeQuietly(…): 这一类方法不仅能在 Connection、Statement 和 ResultSet 为 NULL 情况下避免关闭，还能隐藏一些在程序中抛出的 SQLException。
  - public static void commitAndCloseQuietly(Connection conn)： 用来提交连接，然后关闭连接，并且在关闭连接时不抛出 SQL 异常。
  - public static boolean loadDriver(java.lang.String driverClassName)：这一方装载并注册 JDBC 驱动程序，如果成功就返回 true。使用该方法，你不需要捕捉这个异常 ClassNotFoundException。

QueryRunner

- 该类简单化了 SQL 查询，它与 ResultSetHandler 组合在一起使用可以完成大部分的数据库操作，能够大大减少编码量。
- QueryRunner 类提供了两个构造方法：
  - 默认的构造方法(手动管理事务)
  - 需要一个 javax.sql.DataSource 来作参数的构造方法。（自动管理事物）
- 更新操作
  - public int update(Connection conn, String sql, Object... params)
  - public int update(String sql, Object... params)
- 查询操作
  - public Object query(Connection conn, String sql, ResultSetHandler<T> rsh, Object... params)
  - public Object query(String sql, ResultSetHandler<T> rsh, Object... params)

ResultSetHandler

- 该接口用于处理 java.sql.ResultSet，将数据按要求转换为另一种形式。
- ResultSetHandler 接口提供了一个单独的方法：Object handle (java.sql.ResultSet .rs)。

ResultSetHandler 在 DBUtils 框架中提供九个默认 实现类，直接使用九个默认实现类，可以完成常规操作，而不需要自定义结果集封装

1) ArrayHandler 和 ArrayListHandler 将数据表的每行记录保存 Object[] 中
2) BeanHandler 和 BeanListHandler 将数据表每行记录 保存 JavaBean 对象中
* 封装 javabean 属性时，必须保证数据表列名与 javabean 属性名一致，否则无法封装
3) MapHandler 和 MapListHandler 将结果每行记录保存到一个 Map 集合，key 是列名，value 是值
4) ColumnListHandler 查询结果集中指定一列数据
5) KeyedHandler(name) 结果集每行数据封装 map，再将 map 存入另一个 map 作为 value，指定一列作为 key
6) ScalarHandler 进行单值查询 select count(*) from account;

案例（客户信息管理系统）

字段名	说明	类型
Id	**编号**	**varchar(40)**
name	**客户姓名**	**varchar(20)**
gender	**性别**	**varchar(10)**
birthday	**生日**	**date**
cellphone	**手机**	**varchar(20)**
email	**电子邮件**	**varchar(40)**
preference	**客户爱好**	**varchar(100)**
type	**客户类型**	**varchar(40)**
description	**备注**	**varchar(255)**

```
create table customer(
    id varchar(40) primary key,
    name varchar(20),
    gender varchar(10),
    birthday date,
    cellphone varchar(20),
    email varchar(40),
```

```
    preference varchar(100),
    type varchar(40),
    description varchar(255)
);
```

navicatlite 使用

1、创建 Connection 数据库连接
连接名称可以随意，填写密码就可以了
2、创建数据库 new database
3、创建数据表 new Table

搭建 web project 环境

JavaEE 三层结构
Servlet + JSP + JavaBean+jstl + DBUtils+ DAO + MySQL

导入 jar 包 ： JSTL 、BeanUtils、DBUtils、C3P0、mysql 驱动

创建包结构
cn.itcast.customer.web 表现层
cn.itcast.customer.service 业务层
cn.itcast.customer.dao 持久层
cn.itcast.customer.utils 工具包
cn.itcast.customer.domain 实体类 javaBean

应用的 jar 文件

1. mysql 驱动包
2. c3po 包
3. dbutils 包
4. BeanUtil 包
5. JSTL 包
6. c3p0 的配置文件

编写公共程序 domain utils
Customer 类 实体类
DataSourceUtils 工具类

第十四天: JDBC 分页/条件查询/监听器

分页

- 物理分页
  - 在 sql 查询时，从数据库只检索分页需要的数据
  - 通常不同的数据库有着不同的物理分页语句
  - mysql 物理分页，采用 limit 关键字
  - 例如：检索 11-20 条 select * from user limit 10,10 ;
- 逻辑分页
  - 在 sql 查询时，先从数据库检索出所有数据的结果集
  - 在程序内，通过逻辑语句获得分页需要的的数据
  - 例如: 检索 11-20 条 userList.subList(10,20);

条件查询

- 条件查询 指对数据的查询结果设计条件
  - 一般条件 可以由用户选择

  使用模糊查询 like

监听器

- 监听器就是一个实现特定接口的普通 java 程序，这个程序专门用于监听另一个 java 对象的方法调用或属性改变，当被监听对象发生上述事件后，监听器某个方法将立即被执行。

Servlet 监听器

- 在 Servlet 规范中定义了多种类型的监听器，它们用于监听的事件源分别为 **ServletContext**, **HttpSession** 和 **ServletRequest** 这三个域对象。

- Servlet 规范针对这三个对象上的操作，又把这多种类型的监听器划分为三种类型。
  - 监听三个域对象创建和销毁的事件监听器
  - 监听域对象中属性的增加和删除的事件监听器
  - 监听绑定到 HttpSession 域中的某个对象的状态的事件监听器。（查看 API 文档）

编写监听器

- 和编写其它事件监听器一样，编写 servlet 监听器也需要实现一个特定的接口，并针对相应动作覆盖接口中的相应方法。
- 和其它事件监听器略有不同的是，servlet 监听器的注册不是直接注册在事件源上，而是由 WEB 容器负责注册，开发人员只需在 web.xml 文件中使用<listener>标签配置好监听器，web 容器就会自动把监听器注册到事件源中。
- 一个 web.xml 文件中可以配置多个 Servlet 事件监听器，web 服务器按照它们在

web.xml 文件中的注册顺序来加载和注册这些 Serlvet 事件监听器。

ServletContext 监听器

- ServletContextListener 接口用于监听 ServletContext 对象的创建和销毁事件。
- 当 ServletContext 对象被创建时，激发 contextInitialized (ServletContextEvent sce)方法
- 当 ServletContext 对象被销毁时，激发 contextDestroyed(ServletContextEvent sce)方法。

- 提问，servletContext 域对象何时创建和销毁：
  - 创建：服务器启动针对每一个 web 应用创建 servletcontext
  - 销毁：服务器关闭前先关闭代表每一个 web 应用的 servletContext

示例 1 简单任务调度 Timer 与 TimerTask 使用

HttpSession 监听器

- HttpSessionListener 接口用于监听 HttpSession 的创建和销毁
- 创建一个 Session 时，sessionCreated(HttpSessionEvent se) 方法将会被调用。
- 销毁一个 Session 时，sessionDestroyed (HttpSessionEvent se) 方法将会被调用。

- (此处复习 session 对象，写多个 servlet 都去 getSession，看 session 的创建)
- Session 域对象创建和销毁的时机创建：用户每一次访问时，服务器创建 session
  - 销毁：如果用户的 session 30 分钟没有使用，服务器就会销毁 session，我们在 web.xml 里面也可以配置 session 失效时间

ServletRequest 监听器

- ServletRequestListener 接口用于监听 ServletRequest 对象的创建和销毁。
- Request 对象被创建时，监听器的 requestInitialized 方法将会被调用。
- Request 对象被销毁时，监听器的 requestDestroyed 方法将会被调用。

- (此处复习 request 对象，在浏览器窗口中多次刷新访问 servlet，看 request 对象的创建和销毁，并写一个 servlet，然后用 sendRedirect、forward 方式跳转到其它 servlet，查看 request 对象的创建和消耗)
- servletRequest 域对象创建和销毁的时机：

    - 创建：用户每一次访问，都会创建一个 reqeust
    - 销毁：当前访问结束，request 对象就会销毁

- 当向被监听器对象中增加一个属性时，web 容器就调用事件监听器的 attributeAdded 方法进行相应，这个方法接受一个事件类型的参数，监听器可以通过这个参数来获得正在增加属性的域对象和被保存到域中的属性对象

- 各个域属性监听器中的完整语法定义为：
    - public void attributeAdded(ServletContextAttributeEvent scae)
    - public void attributeAdded (HttpSessionBindingEvent hsbe)
    - public void attributeAdded(ServletRequestAttributeEvent srae)

监听三个域的属性变化

- Servlet 规范定义了监听 ServletContext, HttpSession, HttpServletRequest 这三个对象中的属性变更信息事件的监听器。
- 这三个监听器接口分别是 ServletContextAttributeListener, HttpSessionAttributeListener ServletRequestAttributeListener
- 这三个接口中都定义了三个方法来处理被监听对象中的属性的增加，删除和替换的事件，同一个事件在这三个接口中对应的方法名称完全相同，只是接受的参数类型不同。

- 当向被监听器对象中增加一个属性时，web 容器就调用事件监听器的 attributeAdded 方法进行相应，这个方法接受一个事件类型的参数，监听器可以通过这个参数来获得正在增加属性的域对象和被保存到域中的属性对象

- 各个域属性监听器中的完整语法定义为：
    - public void attributeAdded(ServletContextAttributeEvent scae)
    - public void attributeAdded (HttpSessionBindingEvent hsbe)
    - public void attributeAdded(ServletRequestAttributeEvent srae)

- 当删除被监听对象中的一个属性时，web 容器调用事件监听器的这个方法进行相应
- 各个域属性监听器中的完整语法定义为：
    - **public void attributeRemoved(ServletContextAttributeEvent scae)**
    - **public void attributeRemoved (HttpSessionBindingEvent hsbe)**
    - **public void attributeRemoved (ServletRequestAttributeEvent srae)**

- 当监听器的域对象中的某个属性被替换时，web 容器调用事件监听器的这个方法进行相应
- 各个域属性监听器中的完整语法定义为：
    - **public void attributeReplaced(ServletContextAttributeEvent scae)**
    - **public void attributeReplaced (HttpSessionBindingEvent hsbe)**
    - **public void attributeReplaced (ServletRequestAttributeEvent srae)**

Session 绑定监听器

- 保存在 Session 域中的对象可以有多种状态：绑定到 Session 中；从 Session 域中解除绑定；随 Session 对象持久化到一个存储设备中(钝化)；随 Session 对象从一个存储设备中恢复（活化）
- Servlet 规范中定义了两个特殊的监听器接口来**帮助 JavaBean 对象了解自己在 Session 域中的这些状态：** HttpSessionBindingListener 接口和 HttpSessionActivationListener 接口 ，实现这两个接口的类**不需要 web.xml 文件中进行注册**

- **实现了 HttpSessionBindingListener 接口的 JavaBean 对象可以感知自己被绑定到 Session 中和从 Session 中删除的事件**
- **当对象被绑定到 HttpSession 对象中时，web 服务器调用该对象的 void valueBound(HttpSessionBindingEvent event) 方法**
- **当对象从 HttpSession 对象中解除绑定时，web 服务器调用该对象的 void valueUnbound(HttpSessionBindingEvent event)方法**

- **实现了 HttpSessionActivationListener 接口的 JavaBean 对象可以感知自己被活化和钝化的事件**
- **当绑定到 HttpSession 对象中的对象将要随 HttpSession 对象被钝化之前，web 服务器调用如下方法 sessionWillPassivate(HttpSessionBindingEvent event) 方法**
- **当绑定到 HttpSession 对象中的对象将要随 HttpSession 对象被活化之后，web 服务器调用该对象的 void sessionDidActive(HttpSessionBindingEvent event)方法**

```
<Context>
<Manager className="org.apache.catalina.session.PersistentManager"
maxIdleSwap="1">
<Store className="org.apache.catalina.session.FileStore" directory="it315"/>
</Manager>
</Context>
```

示例 2 在线用户列表与踢人功能

第十五天: javaWeb 之过滤器

Fileter 介绍

- Filter 也称之为过滤器，它是 Servlet 技术中最实用的技术，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
- 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(拦截 url)进行拦截后，WEB 服务器每次在调用 web 资源之前，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

开发 Fileter 步骤

- Filter 开发分为二个步骤：
    - 编写 java 类实现 Filter 接口，并实现(三个方法)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。
- Filter 链 --- FilterChain
    - 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<mapping>，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）

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 方法进行初始化（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)

- **doFilter(ServletRequest,ServletResponse,FilterChain)**
  - 每次 filter 进行拦截都会执行
  - 在实际开发中方法中参数 request 和 response 通常转换为 **HttpServletRequest** 和 **HttpServletResponse** 类型进行操作
- **destroy()：**
  - 在 Web 容器卸载 Filter 对象之前被调用。

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。

注册与映射 Filter

- 注册

```
<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。
- 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。**

```
<filter-mapping>
     <filter-name>testFilter</filter-name>
    <url-pattern>/index.jsp</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

Filter 示例

示例 1 全站统一字符编码过滤器

编写 jsp 输入用户名，在 Servlet 中获取用户名，将用户名输出到浏览器上

处理请求 post 乱码代码
request.setCharacterEncoding("utf-8");
设置响应编码集代码
response.setContentType("text/html;charset=utf-8");

经常会使用，而过滤器可以在目标资源之前执行，将很多程序中处理乱码公共代码，提取到过滤器中 ，以后程序中不需要处理编码问题了

示例 2 禁用所有 JSP 页面缓存

因为动态页面数据，是由程序生成的，所以如果有缓存，就会发生，客户端查看数据不是最新数据情况 ，对于动态程序生成页面，设置浏览器端禁止缓存页面内容

```
response.setDateHeader("Expires",-1);
response.setHeader("Cache-Control","no-cache");
response.setHeader("Pragma","no-cache");
```

将禁用缓存代码，提起到过滤器中，通过 url 配置，禁用所有 JSP 页面的缓存

示例 3 设置图片过期时间

Tomcat 缓存策略

对于服务器端经常不变化文件，设置客户端缓存时间，在客户端资源缓存时间到期之前，就不会去访问服务器获取该资源 -------- 比 tomcat 内置缓存策略更优手段
* 减少服务器请求次数，提升性能

设置静态资源缓存时间，需要设置 Expires 过期时间 ，在客户端资源没有过期之前，不会产生对该资源的请求的
* 设置 Expires 通常使用 response.setDateHeader 进行设置 设置毫秒值

示例 4 自动登录案例（MD5 加密）

在访问一个站点，登陆时勾选自动登陆（三个月内不用登陆），操作系统后，关闭浏览器；过几天再次访问该站点时，直接进行登陆后状态

在数据库中创建 user 表

```
create table user (
    id int primary key auto_increment,
    username varchar(20),
    password varchar(40),
    role varchar(10)
);

insert into user values(null,'admin','123','admin');
insert into user values(null,'aaa','123','user');
insert into user values(null,'bbb','123','user');
```

自动登陆 ：未登录、存在自动登陆信息、自动登陆信息正确

在用户完成登陆后，勾选自动登陆复选框，服务器端将用户名和密码 以 Cookie 形式，保存在客户端 。当用户下次访问该站点，AutoLoginFilter 过滤器从 Cookie 中获取 自动登陆信息
1、判断用户是否已经登陆，如果已经登陆，没有自动登陆的必要

2、判断 Cookie 中是否含有自动登陆信息 ，如果没有，无法完成自动登陆
3、使用 cookie 用户名和密码 完成自动登陆

如果将用户密码保存在 cookie 文件中，非常不安全的 ，通常情况下密码需要加密后才能保存到客户端
* 使用 md5 算法对密码进行加密
* md5 加密算法是一个单向加密算法 ，支持明文---密文 不支持密文解密

MySQL 数据库中提供 md5 函数，可以完成 md5 加密
mysql> select md5('123');
+----------------------------------+
| md5('123') |
+----------------------------------+
| 202cb962ac59075b964b07152d234b70 |
+----------------------------------+
加密后结果是 32 位数字 16 进制表示

Java 中提供类 MessageDigest 完成 MD5 加密

将数据表中所有密码 变为密文 update user set password = md5(password) ;
在登陆逻辑中，对密码进行 md5 加密
在 AutoLoginFilter 因为从 Cookie 中获得就是加密后密码，所以登陆时无需再次加密

示例 5 URL 级别的权限控制

认证：who are you？用户身份的识别 ---------- 登陆功能
权限：以认证为基础 what can you do？您能做什么？ 必须先登陆，才有身份，有了身份，才能确定可以执行哪些操作

示例 6 通用 get 和 post 乱码过滤器

Decorator 模式
1、包装类需要和被包装对象 实现相同接口，或者继承相同父类
2、包装类需要持有 被包装对象的引用
在包装类中定义成员变量，通过包装类构造方法，传入被包装对象
3、在包装类中，可以控制原来那些方法需要加强
不需要加强 ，调用被包装对象的方法
需要加强，编写增强代码逻辑

ServletRequestWrapper 和 HttpServletRequestWrapper 提供对 request 对象进行包装的方法，但是默认情况下每个方法都是调用原来 request 对象的方法，也就是说包装类并没有对

request 进行增强

在这两个包装类基础上，继承 HttpServletRequestWrapper ，覆盖需要增强的方法即可
系统中存在很多资源，将需要进行权限控制的资源，放入特殊路径中，编写过滤器管理访问特殊路径的请求，如果没有相应身份和权限，控制无法访问

在 Filter 中，对 request 对象进行包装，增强获得参数的方法
getParameter
getParameterValues
getParameterMap

ServletResponseWrapper 和 HttpServletResponseWrapper 提供了对 response 对象包装，继承 HttpServletResponseWrapper ，覆盖需要增强 response 的方法

示例 7 压缩响应案例

Tomcat 服务器内，提供对响应压缩 配置实现
在 conf/server.xml 中
<Connector port="80" protocol="HTTP/1.1"
connectionTimeout="20000"
redirectPort="8443"/> 添加 compressableMimeType 和 compression
没有压缩 ： 00:00:00.000 0.063 7553 GET200 text/html http://localhost/

<Connector port="80" protocol="HTTP/1.1"
connectionTimeout="20000"
redirectPort="8443" compressableMimeType="text/html,text/xml,text/plain" compression="on"/>

压缩后 ： 00:00:00.000 0.171 2715 GET200 text/html http://localhost/
Content-Encoding: gzip
Content-Length : 2715

实际开发中，很多情况下，没有权限配置 server.xml ,无法通过 tomcat 配置开启 gzip 压缩

编写过滤器对响应数据进行 gzip 压缩

flush 方法
只有没有缓冲区字节流，FileOutputStream 不需要 flush
而字节数组 ByteArrayOutputStream、字节包装流、字符流 需要 flush ----- 这些流在调用 close 方法时都会自动 flush

第十六天 文件上传与下载

上传

文件上传概述

- 实现 web 开发中的文件上传功能，需完成如下二步操作：
  - 在 web 页面中添加上传输入项
  - 在 servlet 中读取上传文件的数据，并保存到服务器硬盘中。
- 如何在 web 页面中添加上传输入项?
  - <input type=“file”>标签用于在 web 页面中添加文件上传输入项，设置文件上传输入项时须注意：
  - 1、必须要设置 input 输入项的 name 属性，否则浏览器将不会发送上传文件的数据。
  - ２、必须把 form 的 enctype 属值设为 **multipart/form-data.**设置该值后，浏览器在上传文件时，将把文件数据附带在 http 请求消息体中，并使用ＭＩＭＥ协议对上传的文件进行描述，以方便接收方对上传数据进行解析和处理。
  - 3、表单的提交方式要是 post
- 如何在 Servlet 中读取文件上传数据，并保存到本地硬盘中?
  - Request 对象提供了一个 getInputStream 方法，通过这个方法可以读取到客户端提交过来的数据。但由于用户可能会同时上传多个文件，在 servlet 端编程直接读取上传数据，并分别解析出相应的文件数据是一项非常麻烦的工作，示例。
  - 为方便用户处理文件上传数据，Apache 开源组织提供了一个用来处理表单文件上传的一个开源组件（ Commons-fileupload ），该组件性能优异，并且其 API 使用极其简单，可以让开发人员轻松实现 web 文件上传功能，因此在 web 开发中实现文件上传功能，通常使用 Commons-fileupload 组件实现。
- 使用 Commons-fileupload 组件实现文件上传，需要导入该组件相应的支撑 jar 包：Commons-fileupload 和 commons-io。commons-io 不属于文件上传组件的开发 jar 文件，但 Commons-fileupload 组件从 1.1 版本开始，它工作时需要 commons-io 包的支持。

文件上传步骤

- **实现步骤**

１、创建 DiskFileItemFactory 对象，设置缓冲区大小和临时文件目录
２、使用 DiskFileItemFactory 对象创建 ServletFileUpload 对象，并设置上传文件的大小限制。
３、调用 ServletFileUpload.parseRequest 方法解析 request 对象，得到一个保存了所有上传内容的 List 对象。
４、对 list 进行迭代，每迭代一个 FileItem 对象，调用其 isFormField 方法判断是否是上传文件

- True 为普通表单字段，则调用 getFieldName、getString 方法得到字段名和字段值
- False 为上传文件，则调用 getInputStream 方法得到数据输入流，从而读取上传数据。

FileUpload 上传操作核心 API

1、DiskFileItemFactory 磁盘文件项工厂类
public DiskFileItemFactory(int sizeThreshold, java.io.File repository) 构造工厂时，指定内存缓冲区大小和临时文件存放位置

public void setSizeThreshold(int sizeThreshold) 设置内存缓冲区大小，默认 10K

public void setRepository(java.io.File repository) 设置临时文件存放位置，默认 System.getProperty("java.io.tmpdir").

内存缓冲区： 上传文件时，上传文件的内容优先保存在内存缓冲区中，当上传文件大小超过缓冲区大小，就会在服务器端产生临时文件
临时文件存放位置： 保存超过了内存缓冲区大小上传文件而产生临时文件
* 产生临时文件可以通过 FileItem 的 delete 方法删除

2、ServletFileUpload 文件上传核心类
static boolean isMultipartContent(javax.servlet.http.HttpServletRequest request) 判断 request 的编码方式是否为 multipart/form-data

java.util.List parseRequest(javax.servlet.http.HttpServletRequest request) 解析 request，将请求体每个部分封装 FileItem 对象，返回 List<FileItem>

void setFileSizeMax(long fileSizeMax) 设置单个文件上传大小
void setSizeMax(long sizeMax) 设置总文件上传大小

void setHeaderEncoding(java.lang.String encoding) 设置编码集 解决上传文件名乱码 *****

3、FileItem 表示文件上传表单中 每个数据部分
boolean isFormField() 判断该数据项是否为文件上传项，true 不是文件上传 false 是文件上传
```
if(fileItem.isFormField()){
    // 不是上传项
    java.lang.String getFieldName() 获得普通表单项 name 属性
    java.lang.String getString() / java.lang.String getString(java.lang.String encoding) 获得普通表单项 value 属性 传入编码集用来解决输入 value 乱码
}else{
    // 是上传项
```

java.lang.String getName() 获得上传文件名 （注意 IE6 存在路径）
java.io.InputStream getInputStream() 获得上传文件内容输入流
// 上传文件
void delete() 删除临时文件（删除时，必须要管理输入输出流）
}

注意事项：因为文件上传表单采用编码方式 multipart/form-data 与传统 url 编码不同，所有 getParameter 方法不能使用 setCharacterEncoding 无法解决输入项乱码问题

JavaScript 的多文件上传表单

- 技巧：
  - 每次动态增加一个文件上传输入框，都把它和删除按纽放置在一个单独的 div 中，并对删除按纽的 onclick 事件进行响应，使之删除删除按纽所在的 div。
  - 如：
    this.parentNode.parentNode.removeChild(this.parentNode);

上传文件存在的问题

➢ 上传文件后，在服务器端保存位置

第一类存放位置：直接存放 WebRoot 目录下 和 除 WEB-INF META-INF 的其它子目录下
例如: WebRoot/upload
* 客户端可以直接在浏览器上通过 url 访问位置（资料无需通过权限控制，而可以直接访问）
---- 对上传资源安全性要求不高、或者资源需要用户直接可见
* 例如：购物商城商品图片

第二类存放位置：放入 WEB-INF 及其子目录 或者 不受 tomcat 服务器管理目录 例如：WebRoot/WEB-INF/upload 、c:\ 、d:\abc
* 客户端无法通过 URL 直接访问，必须由服务器内部程序才能读取 （安全性较高，可以很容易添加权限控制）
* 例如：会员制在线视频

➢ 上传文件在同一个目录重名问题

如果文件重名，后上传文件就会覆盖先上传文件

文件名 UUID
filename = UUID.randomUUID().toString() + "_" + filename;

➢ 为了防止同一个目录下方上传文件数量过多 ---- 必须采用目录分离算法

1) 按照上传时间进行目录分离 （周、月 ）

2) 按照上传用户进行目录分离 ----- 为每个用户建立单独目录
3) 按照固定数量进行目录分离 ------ 假设每个目录只能存放 3000 个文件 ，每当一个目录存满 3000 个文件后，创建一个新的目录
4) 按照唯一文件名的 hashcode 进行目录分离

```
public static String generateRandomDir(String uuidFileName) {
    // 获得唯一文件名的 hashcode
    int hashcode = uuidFileName.hashCode();
    // 获得一级目录
    int d1 = hashcode & 0xf;
    // 获得二级目录
    int d2 = (hashcode >>> 4) & 0xf;

    return "/" + d2 + "/" + d1;// 共有 256 目录
}
```

- 乱码问题

普通编写项 value 属性乱码 ---------- fileItem.getString(编码集);
上传文件项 文件名乱码 --------- fileupload.setHeaderEncoding(编码集);

下载

常见文件下载有两种方式

1、超链接直接指向下载资源
如果文件格式浏览器识别，将直接打开文件，显示在浏览器上， 如果文件格式浏览器不识别，将弹出下载窗口
对于浏览器识别格式的文件，通过另存为进行下载

客户端访问服务器静态资源文件时，静态资源文件是通过 缺省 Servlet 返回的，在 tomcat 配置文件 conf/web.xml 找到 --- org.apache.catalina.servlets.DefaultServlet

2、编写服务器程序，读取服务器端文件，完成下载
必须设置两个头信息 ，来自 MIME 协议 Content-Type Content-Disposition

```
response.setContentType(getServletContext().getMimeType(filename));
response.setHeader("Content-Disposition", "attachment;filename=" + filename); // 以附件形式打开，不管格式浏览器是否识别
```

处理 IE 浏览器与 Firefox 浏览器乱码问题

```
if (agent.contains("MSIE")) {
```

```
            // IE 浏览器
            filename = URLEncoder.encode(filename, "utf-8");
            filename = filename.replace("+", " ");
        } else if (agent.contains("Firefox")) {
            //  火狐浏览器
            BASE64Encoder base64Encoder = new BASE64Encoder();
            filename = "=?utf-8?B?"
                    + base64Encoder.encode(filename.getBytes("utf-8"))
                    + "?=";
        } else if (agent.contains("Chrome")) {
            // google 浏览器
            filename = URLEncoder.encode(filename, "utf-8");
        } else {
            //  其它浏览器
            filename = URLEncoder.encode(filename, "utf-8");
        }
```

综合案例 网盘系统

需求：
1、系统提供一个文件上传功能，在用户上传文件后，文件保存在服务器端指定目录，文件相关信息保存在数据库中
* 每上传一个文件，数据库中存在一条数据记录
2、系统提供一个文件下载功能，将数据表中所有资源信息，显示在页面上，允许用户进行下载

创建数据库环境

```
create database day16

create table resources(
   id int primary key auto_increment,
   uuidname varchar(100) unique not null,
   realname varchar(40) not null,
   savepath varchar(100) not null,
   uploadtime timestamp ,
   description varchar(255)
);
```

导入 jar 包 、c3p0-config.xml 、JDBCUtils 工具类

第十七天 JavaMail/在线支付

Java 邮件开发

邮件开发相关介绍

1、邮件服务器和电子邮箱
邮件服务器 ： 在互联网中（互联网 ip 地址） 一台安装邮件服务器软件的主机 ---- 收发邮件
电子邮箱 ：邮件服务器上账号 ，邮件服务器会为每个邮箱账户分配一定空间，用来保存电子邮件

2、邮件传输协议
什么是协议 ： 客户端与服务器 进行交互 规范的数据格式
SMTP ： 发送邮件的协议 端口：25
POP3 ： 收取邮件的协议 端口：110

IMAP ： 收取邮件协议，默认端口 143 ---- 支持在线操作
* 它与 POP3 协议的主要区别是用户可以不用把所有的邮件全部下载，可以通过客户端直接对服务器上的邮件进行操作。
* gmail 就支持 IMAP 协议收取邮件

3、电子邮件收发过程
aaa@sina.com 发信给 bbb@sohu.com
1) aaa 登陆邮件客户端程序
2) aaa 所在邮件客户端 连接 sina 的 SMTP 服务器
3) aaa 编写邮件，需要从 sina 的 SMTP 服务器 转投 给 sohu 的 SMTP 服务器
4) sohu 的 SMTP 服务器收到信件后，保存 bbb 账户对应空间中
5) bbb 登陆邮件客户端程序
6) bbb 所在邮件客户端连接 sohu 的 pop3 邮件服务器，读取 bbb 账户空间获得 aaa 发送的邮件

使用 telnet 测试发送与接收邮件

准备：在 sina 和 sohu 分别注册一个邮箱账户
sina : duhong4790@sina.com / duhong
sohu : duhong4790@sohu.com / duhong

用新浪的账户向搜狐账户发送一封邮件
安装 telnet

cmd> telnet smtp.sina.com 25

ehlo xxx
auth login
ZHVob25nNDc5MA==
ZHVob25n
mail from:<duhong4790@sina.com>
rcpt to:<duhong4790@sohu.com>
data ------ 发送邮件必须遵循 RFC822 文档规范
from:<duhong4790@sina.com>
to:<duhong4790@sohu.com>
subject: 测试邮件

这是一封测试邮件!
.
quit

抄送： A 发送邮件给 B ，需要同时发送邮件给 C （邮件主要是发给 B 的，需要 C 知道，选择抄送给 C ， 抄送时 B 可以看到的 ）
暗送： A 发送邮件给 B ，需要发送给 C ，但是不想让 B 知道，选择暗送给 C （暗送接收方无法得知）

使用 pop3 协议收信
cmd> telnet pop3.sohu.com 110
user duhong4790
pass duhong

list 查看邮件列表信息
stat 邮件统计信息
retr 收取指定邮件

quit 退出

安装易邮邮件服务器

修改 工具 --- 服务器配置 estore.com
新建两个账户 aaa/111 bbb/111 账户 - 新建账户

2、采用手工方式用 aaa@estore.com 向 bbb@estore.com 发送一封邮件
cmd > telnet localhost 25
ehlo xxx
auth login
YWFh
MTEx

mail from:<aaa@estore.com>
rcpt to:<bbb@estore.com>
data
from:<aaa@estore.com>
to:<bbb@estore.com>
subject: 测试邮件

这是一封测试邮件！

.
quit

3、采用手工方式 bbb 进行收信
cmd> telnet localhost 110
user bbb
pass 111
retr 2 ----- 收信
quit

window7 手动 smtp 发送邮件，中文输入后显示 ？？ ---- telnet 客户端导致的
putty 小巧客户端工具的使用

putty 实现 telnet 功能，而且支持很多其他连接协议 ，功能非常强大，常用来在企业中远程操作 Linux 系统

配置 putty
1、window translation 设置接收数据字符集
默认 Use font encoding 等价于 gbk
2、window appearance 设置显示字符集 change 新宋体 字符集 gb2312
3、Session
主机 ip : localhost
port : 25
协议 ： telnet

Outlook 与 foxmail 介绍

- Outlook

outlook 软件是微软提供专门收发邮件客户端软件，是 office 套件之一，收费软件
win7 支持版本 2007 以上 xp 支持 2003

第一次使用 outlook 直接进行配置
工具 -- 电子邮件账户 --- 查看账户 -- 添加
1) 选取接收邮件服务器类型 pop3

2) 姓名随意，邮件地址填写邮件账户 ：bbb@estore.com
3) 填写密码 111

4) 编写接收服务器 pop3 地址 localhost ---------- pop.sina.com
5) 发送邮件服务器 smtp 地址 localhost ---------- smtp.sina.com
6) 其它设置 -- 发送服务器 --- 勾选我的发送服务器 SMTP 要求验证
7) 其它设置 --- 高级 --- 勾选在服务器上保留邮件副本

win7 系统 ，因为采用 IPV6 地址，无法连接本地 易邮 ---- 尝试在 hosts 文件中添加 127.0.0.1 localhost 能否解决

- Foxmail

foxmail 免费邮件客户端

安装 foxmail 后
第一次启动应该需要配置
邮箱 -- 新建邮箱账户

1) 电子邮箱地址 bbb@estore.com 密码 111
2) pop3: localhost
smtp : localhost

foxmail 默认 smtp 需要验证，自动保存邮件副本

右键邮箱 属性 --- 接收邮件 --- 勾选每隔 15 分钟自动接收邮件

RFC822 文档

RFC822 文档 和 SMTP 协议 有什么区别 ？ 相当于 HTML 和 HTTP
RFC822 文档 规定邮件内容格式
SMTP 发送邮件协议 ，传输数据格式

问题：to 、cc 和 bcc 区别 ？

RFC822 文档漏洞 ---- 冒名邮件问题 mail from 与 邮件正文中 from 内容不一致，收件人只能看到 from 的内容，无法得知真正发件人是谁

cmd> telnet localhost 25

ehlo xxx
auth login
YWFh
MTEx
mail from:<aaa@estore.com>
rcpt to:<bbb@estore.com>

data
from:<hr@google.com>
to:<bbb@estore.com>
subject:面试通知

尊敬的 XX 先生：
　　我公司近期从 51JOB 上看到您的简历，发现与我公司正在招聘高级 Java 软件工程师职位非常符合，请于 2013 年 8 月 1 日 8:00 到苏州街银科大厦 B 座 1201 室，参加面试！

google 人力资源部
2013-08-01
.
quit

Java 开发邮件快速入门

JavaMail 是 sun 公司提供 官方一套用于收发邮件 API 类库 ---- 使用 JavaMail 主要编写邮件客户端程序，功能类似 outlook foxmail

1、下载 JavaMail 的 jar 包
需要下载 javamail 和 jaf 的 jar 包 ----- javamail API 依赖于 jaf
JAF 是一个专用的数据处理框架，它用于封装数据，并为应用程序提供访问和操作数据的接口

2、在工程中导入 javamail 的 jar 包
jaf 从 JDK6.0 开始，内置了 ----- javax.activation 包
如果使用 JDK6.0 以后版本开发，只需要导入 javamail.jar 就可以了
如果使用 JDK5.0 之前开发，需要同时导入 javamail 和 jaf 的 jar 包

将 mail.jar 复制 WEB-INF/lib 目录

3、使用 JavaMail 开发邮件
Session 表示与邮件服务器连接会话
Message 表示一封邮件

使用 MyEclipse 开发 javamail 程序 异常 ： java.lang.NoClassDefFoundError: com/sun/mail/util/LineInputStream
原因 ： myeclipse 自带 javaee 的 jar 包 与 javamail 的 jar 冲突
解决 ： 删除 javaee.jar 中 javax.activation 和 javax.mail 两个包

window -- preferences -- 搜索 lib 选中 myeclipse 下 library sets --- JavaEE5.0
选中 javaee.jar --- add jar --- 右键选中 jar 使用压缩工具打开 --- 删除上面两个包

4、将邮件发送出去

JavaMail 收发邮件围绕四个核心类进行
1) Session 与服务器连接会话
2) Message 邮件内容
3) Transport 发送邮件核心工具类
4) Store 接收邮件核心工具类

发送邮件步骤：
1、建立与邮件服务器会话连接 Session --- Appendix A 附录 A
2、编写邮件
3、发送

MIME 协议

1、MIME 协议
MIME 协议是对 RFC822 文档的升级补充，在 MIME 协议中描述了如何编写一封复杂邮件
如果一封邮件遵循 RFC822 文档编写，通用符合 MIME 协议格式的，MIME 协议兼容 RFC822 文档

MIME 将邮件中各个数据部分，用分隔线分隔，为每个数据部分提供 Content-Type 字段 ----- 表示该部分数据类型

各部分之间数据关系 有三种
1) multipart/mixed 用于携带附件
2) multipart/related 内嵌图片，音乐
3) multipart/alternative 防止兼容问题
multipart/alternative 指的是，邮件中同时存在两种数据格式，例如一种文本内容，另一种 HTML 内容，设置关系 multipart/alternative ，如果邮件客户端支持 HTML 显示，显示 HTML 内容，如果不支持 HTML 显示，显示文本内容

Content-ID 字段：用于为内嵌图片设置 唯一编号，设置编号后，在正文中通过该编号，引用这张图片
*

Content-Disposition 字段：用于指定附件，以附件查看方式打开
* Content-Disposition: attachment;filename=xxx

2、MIME 协议的编码

From: aaa@estore.com
To: bbb@estore.com
Message-ID: <11025290.0.1352183585703.JavaMail.seawind@PC2011101416ajs>
Subject: =?GBK?Q?javamail=B7=A2=CB=CD=BC=F2=B5=A5=D3=CA=BC=FE?=
MIME-Version: 1.0
Content-Type: text/plain; charset=GBK

Content-Transfer-Encoding: base64

vPK1pdPKvP7V/c7ExNrI3SAuLi4=

MIME 协议支持两种编码：BASE64 和 QP
邮件标题采用 QP 编码
邮件正文采用 BASE64 编码

JavaMail 中提供了对数据进行 BASE64 和 QP 编码 API： MimeUtility 工具类

3、JavaMail API 中如何定义复杂邮件的
MimeMessage 代表一封符合 MIME 协议邮件
MimeMultipart 多个数据部分组合
MimeBodyPart 代表复杂邮件中一个数据部分

MimeMessage 可以由一个 MimeMultipart 构成内容
MimeMultipart 整合多个 MimeBodyPart
MimeMultipart 必须包装为 MimeBodyPart 才可以与其他 MimeBodyPart 进行整合

Java 发送复杂邮件

- 发送一封内嵌图片邮件
- 发送一封携带附件的邮件

在线支付

在线支付方式

方案一：网站直接与银行对接 ，需要网站针对不同银行接入规范，编写不同银行接入程序
缺点：不同银行存在不同接入规范，网站开发维护人员工作量极大 ，如果银行接口规范变动，网站程序必须要进行修改

方案二：网站只负责与第三方支付公司对接，由支付公司与不同银行完成对接
优点：网站开发维护工作量很低，不用关心银行接入规范
缺点：第三方支付公司收取一定手续费用，通常每笔交易 1%；方案一用户付款后 银行直接将钱转给网站，方案二 用户付款后，银行将钱转给支付公司，再由支付公司转给网站

支付流程和原理

身份数据识别问题 ----- 解决方案 ： 数字签名技术
1、商家将支付请求数据 获得后，需要使用支付公司提供密钥，按照一定加密算法（由支付公司提供），对支付的数据进行加密 ，获得 hmac 码（数字签名）
商家将支付的数据（未加密）和 hmac 码 一起发送给支付公司 支付公司使用同样密钥和算法 对支付数据进行加密，获得一个数字签名，比较和商家传来 hmac 是否一致 ，如果 hmac 一致证明，数据来源于商家，证明数据在传输的过程中没有被篡改过。

数据 ＋key＋ 算法 ＝hmac

2、同样的方式，商家收到支付公司付款成功通知后，获得支付公司传来的数据，采用密钥和算法进行加密获得 hmac 码，与支付公司传来的 hmac 码进行比较，如果一致，证明数据来源于支付公司，数据在传递的过程中没有被篡改

在线支付前提

1、网站必须要具有公网 ip 地址
在付款成功后，易宝通知网站付款成功 ，如果网站不能被公网访问，意味着易宝无法通知付款结果

测试公网 ip 可否访问

2、必须要有网银账户

支付成功回调方式分为：浏览器重定向和服务器点对点
1)对于浏览器重定向，显示给用户付款成功！不要更改订单状态
2)对于服务器点对点，在可以更改订单状态，因为请求来自于易宝支付 --- 回复 success

第十八----二十天 Estore 系统

Estore 系统分析

Estore 购物商城项目 （综合练习）
目的：将之前学习知识点 整合，综合运用

系统需求分析

功能：
1、用户注册
2、用户登录
3、添加商品（CURD）
4、商品查看-- 列表查询
5、商品详情页面
6、将商品添加购物车
7、查看购物车
8、修改购物车
9、生成订单
10、订单查看（取消）
11、在线支付
12、销售榜单查看

需求分析 UML 用例图
游客（未登录）： 注册、登陆、商品查看
商城注册用户 ： 商品查看、添加商品到购物车、购物车管理、生成订单、订单管理、在线支付
管理员 ： 添加商品、商品管理、查看订单 、榜单查看（导出）

系统设计

1、技术选型 （系统架构）
JSTL + JSP + Servlet + JavaBean + BeanUtils + FileUpload + JavaMail + DBUtils(JDBC) + C3P0 + MySQL + MyEclipse10+ Tomcat7.0 + JDK6 + Windows
MVC 模式
JavaEE 三层结构
DAO 模式

2、数据库设计 E-R 图
绘制 E-R 图 和 面向对象的需求分析 PowerDesigner (概念图 E-R 、物理图表结构、面向对象图) 三种图相互转换
1) 抽象需求分析中名词 成为实体/类
2) 根据系统需求，分析实体/类 需要哪些属性
3) 建立实体之间关系

系统存在五个实体：用户、商品、订单、购物车、榜单
购物车不放入数据库 （Session 或者 Cookie 实现）
榜单 （待定）

根据 E-R 图/对象关系建表

先创建实体对应表，再描述属性字段、最后描述关系
* E-R 图展示实体属性 不一定包括所有表字段
用户表

```
create table users (
    id int primary key auto_increment,
    username varchar(40),
    password varchar(100),
    nickname varchar(40),
    email varchar(100),
    role varchar(100) ,
    state int ,
    activecode varchar(100),
    updatetime timestamp    );
```

商品表

```
create table products(
    id varchar(100) primary key ,
    name varchar(40),
    price double,
    category varchar(40),
    pnum int ,
    imgurl varchar(100),
    description varchar(255));
```

订单表

```
create table orders(
    id varchar(100) primary key,
    money double,
    receiverinfo varchar(255),
    paystate int,
    ordertime timestamp,
    user_id int ,
    foreign key(user_id) references users(id)
);
```

用户与订单之间存在 一对多关系 ： 在多方添加一方主键作为外键
订单和商品之间存在 多对多关系 ： 创建第三张关系表，引入两张表主键作为外键 （联合主键）
订单项

```
create table orderitem(
    order_id varchar(100),
    product_id varchar(100),
    buynum int ,
    primary key(order_id,product_id),
    foreign key(order_id) references orders(id),
    foreign key(product_id) references products(id)
```

);

设置数据库环境
数据库 ： create database estoresystem;

工程环境搭建和网站部署

1) 导入 jar 包
导入 mysql 驱动 mysql driver / mysql-connector-java-5.0.8-bin.jar
导入 c3p0 c3p0/c3p0-0.9.1.2.jar 将 c3p0-config.xml 复制 src 下 将 DataSourceUtils 复制 cn.itcast.estore.utils ----- 配置 c3p0-config.xml 数据库连接参数
导入 dbutils apache commons\dbutils\commons-dbutils-1.4.jar
导入 beanutils commons-beanutils-1.8.3.jar commons-logging-1.1.1.jar
导入 fileupload commons-fileupload-1.2.1.jar commons-io-1.4.jar
导入 javamail mail.jar
导入 jstl jstl.jar standard.jar

2) 建立 package 结构 按照 JavaEE 三层结构
cn.itcast.estore.web.servlet
cn.itcast.estore.web.filter
cn.itcast.estore.web.listener
cn.itcast.estore.service
cn.itcast.estore.dao
cn.itcast.estore.domain
cn.itcast.estore.utils

3) domain 类编写
UML 中类图画法

4) 工程发布
将 estore 项目配置虚拟主机，以顶级域名方式进行发布
1) 将工程根目录 estore 目录 配置虚拟主机目录 ---- 配置 conf/server.xml

```
<Host name="www.estore.com"  appBase="myeclipse 下的工程路径"
        unpackWARs="true" autoDeploy="true"
        xmlValidation="false" xmlNamespaceAware="false">
</Host>
```

2) 将工程目录下 WebRoot 目录，配置缺省 web 应用

```
<Host name="www.estore.com"  appBase=" myeclipse 下的工程路径"
        unpackWARs="true" autoDeploy="true"
        xmlValidation="false" xmlNamespaceAware="false">
        <Context path="" docBase="WebRoot" />
</Host>
```

3) 将 www.estore.com 虚拟主机配置缺省虚拟主机 ----- 能够用 ip 直接访问主机

```
<Engine name="Catalina" defaultHost="www.estore.com">
```

4) 修改本机域名解析文件 hosts ---- c:\windows\system32\drivers\etc
添加 127.0.0.1 www.estore.com

用户注册

1、 功能一： 用户注册 （技术点：验证码技术 、激活邮件技术）
1) 一次性验证码
原理：在生成验证码 Servlet 程序中，将生成验证码保存 Session 中，用户提交验证码 与 保存在 Session 验证码进行比较，如果相同，请求合法

2) 注册表单 JS 校验
考虑 JS 校验是否可以抽取成框架
3) 处理 form 乱码问题 ---- 通用 get post 乱码过滤器
4) 密码 MD5 加密

5) 激活邮件发送
启动易邮 域名 estore.com
建立几个账号 service/111 aaa/111 bbb/111 ccc/111 ddd/111

账号激活 ，判断激活码存在，并且有效 ------- update 修改激活 state 字段

6) 配置通用错误处理页面 day8 配置 web.xml

```
<error-page>
   <error-code>500</error-code>
   <location>/500.jsp</location>
</error-page>

<error-page>
   <error-code>404</error-code>
   <location>/404.jsp</location>
</error-page>
```

500.jsp 显示错误的 msg
404.jsp 自动刷新跳转回主页面

用户登录

原理：用户输入正确用户名和密码，登陆成功，用户信息将会被保存 Session 对象中。 ------
记住用户名和密码 、自动登陆
权限控制围绕登陆功能

1) 登陆表单中，添加记住用户名 和 自动登陆功能

2) 登陆过程中判断 账户是否激活

3) 在 login.jsp 显示记住用户名
在 username 的 input 项中添加 value="${cookie.username.value}"
在勾选自动登陆 checkbox 添加 c:if 判断

4) 登陆后注销功能
注销 Session

用户自动登录过滤器

自动登陆功能 对系统所有页面有效 （例如访问 index.jsp list_product.jsp info_product.jsp 这些页面在访问时都将执行自动登陆 ）
* 对于登陆相关页面不会执行自动登陆逻辑 （login.jsp 、LoginServlet 、InvalidateServlet ）
1) 判断该请求页面是否需要自动登陆
2) 是否已经登陆
3) 是否含有自动登陆 cookie
4) 自动登陆

5) 在 LoginServlet 添加对于没有勾选记住用户名和自动登陆 处理代码 !!!!!!!
6) 在退出功能 InvalidateServlet 清除自动登陆信息

商品添加

文件上传三个注意事项
1) input 输入框 必须有 name 属性
2) 表单 form 必须 post 提交方式
3) 设置 form 的 enctype 为 multipart/form-data

表单提交时，校验分为两种 ： 客户端校验 、服务器端校验 ------ 只有服务器端校验才能确保数据准确

商品图片上传到服务器端后，保存在哪个目录 ？？？ 必须直接在 WebRoot 下及其除 WEB-INF、META-INF 子目录外

一般情况下，一张表对应 Domain 类 --- DAO 类 ---- Service 类

商品列表查看

条件查询、分页查询

列表中显示原图，因为原图比较大，页面加载非常缓慢 ，页面布局不会很美观 ----------缩略图
* Java 中通过图形界面技术，生成小图

在 Product 类中添加 getImgurl_s 方法，用来获取缩略图路径

将生成缩略图代码 添加 AddProductServlet 中

商品详细信息查看

通过在列表中点击 商品图片或者商品名称 进入详情查看页面

权限控制过滤器

原理：在过滤器获得来访者资源路径，判断路径需要哪些权限，获取当前登陆用户信息，判断当前登陆用户是否具有需要权限， 如果具有放行，否则抛出权限不足异常
因为系统中只存在三种角色： 未登录（游客）、已登录（管理员、客户）

1、为每类角色定义配置文件，管理员 admin.txt 客户 user.txt ---- 将需要 admin 角色才能访问资源放入 admin.txt 中，将需要 user 角色才能资源 放入 user.txt
2、在 src 下创建 admin.tx 和 user.txt 读取两个文件
getServletContext().getRealPath("/WEB-INF/classes/admin.txt"); // 如果工程路径中有中文和空格，该方法可以使用
XXX.class.getResource("/admin.txt").getFile(); // 如果工程路径中有中文和空格，该方法无法读取到文件 ，空格和中文会进行 URL 编码

在系统中插入 admin 角色
insert into users values(null,'admin',md5('123'),'超级管理员','service@estore.com','admin',1,null,null);

添加商品到购物车

购物车对象，不保存在数据库中，使用 Session 来保存用户购物车数据

保存购物车对象 Map<Product,Integer> key 商品对象 value 商品购买数量

流程 ：添加商品到购物车流程，点击添加到购物车，将商品 id 传递 Servlet ，从 Session 中取出购物车对象，判断商品是否已经在购物车中，如果不在添加商品到购物车数量 1 ，在购物车取出原有数量+1

1) 如果 Map 的 key 是一个自定义对象，重写 hashcode 和 equals

2) 如果商品不在购物车中，需要根据商品 id 查询商品所有信息，添加购物车

3) 添加商品到购物车

Session 中信息在服务器正常关闭时，会被序列化到硬盘中 ---- Product 实现 Serializable 接口 完成序列化

显示与修改购物车数据

购物车信息保存在 Session 中，不需要去查询数据库，将 Session 中信息显示出来

1) 在购物车中显示总价

2) 购物车修改 ：

清空购物车 ： request.getSession().removeAttribute("cart");

删除购物车中单项商品 ： cart.remove(product) ； 注意：删除一项后，判断购物车是否为空，如果为空 移除购物车对象

* 删除确认功能

写法一：直接在 href 中 触发 js 函数，询问用户是否确认，如果确认，location.href 发起删除请求 ----- ...

写法二： 在链接中添加 onclick 事件 ，询问用户是否确认，如果用户取消，通过 JS 阻止 href 事件提交

阻止 href 默认事件 ： e.preventDefault() ---- 必须支持事件 FF 支持、IE 不支持

IE 阻止 href 默认事件

```
function confirmDel(e){
    // 询问用户是否确认
    var isConfirm = window.confirm("商品不要了吗？多好的商品啊！");
    if(!isConfirm){
        // 用户选择取消，阻止 a 标签 默认事件 href 发生
        if(e&&e.preventDefault){
            // e 对象存在，preventDefault 方法存在 ---- 火狐浏览器
            e.preventDefault();
        }else{
            // 不支持 e 对象，或者没有 preventDefault 方法 ---- IE
            window.event.returnValue = false;
        }
    }
}
```

修改购物车中商品购买数量 ---- JavaScript 控制购物数量修改

订单生成

在 listcart.jsp 添加进入结算中心 ---- order.jsp

重点：

1) 向 orders 表插入订单信息后，同时需要向 orderitem 表插入 订单中每项数据

2) 多表插入 （数据完整性问题） --- 事务管理

```
    Connection con = null;
        try {
            con = DataSourceUtils.getConnection();
            con.setAutoCommit(false);

            // 1.insert orders
            OrderDao oderDao = new OrderDao();
            OrderDao.addOrder(con, order);

            // 2.insert orderItem
            ProductDao pdao=new ProductDao();
            List<OrderItem> orderitems = order.getOrderItems();
            for (int i = 0; i < orderitems.size(); i++) {
                oderDao.addOrderItem(con,orderitems.get(i)); //一个订单中
有多个商品

                // 3.update product 数量
                pdao.updateProductPnum(con,orderitems.get(i));

            }

        } catch (Exception e) {
            try {
                DbUtils.rollback(con);
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
            e.printStackTrace();
            throw new RuntimeException("订单生成失败");
        } finally {
            try {
                DbUtils.commitAndClose(con);
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
```

3) 订单生成后 ，商品数量更新减少

订单列表查询与取消

管理查询订单 --- 所有人订单信息
普通用户 ---- 只能查询自己的订单

在订单生成后，查看订单列表 、也可以通过 index.jsp 进入订单查看页面

重点：查询订单时，同时查询订单项目信息
1) 查询订单基本信息时，查询下单用户昵称和用户名
在 Order 类中 添加 用户的 username 和 nickname 字段
select orders.*,users.username,users.nickname from orders,users where orders.user_id = users.id;

2) 查询订单项信息时，查询商品的名称和单价
在 OrderItem 类中 添加商品的 name 和 price 字段
select orderitem.*,products.name,products.price from orderitem,products where orderitem.produtct_id = products.id and orderitem.order_id = ?

订单取消 ： 删除订单表信息时，同时删除订单项信息 （订单项信息依赖 订单信息 ，必须先删除订单项）
注意事项
1) 可以取消未支付的订单，如果订单已经支付，将无法取消
2) 管理员不能取消任何用户未支付订单，普通用户只能取消 自己的未支付的订单

取消订单进行事务管理 ：删除订单项、删除订单、恢复商品数量

订单在线支付

1) 网站进行支付，根据易宝支付接口请求规范，生成易宝需要数据，将数据提交到易宝指定网址 https://www.yeepay.com/app-merchant-proxy/node
* 在提交支付请求给易宝，需要将请求中数据，使用易宝提供密钥和算法进行加密，获得数字签名 hmac 码 ，将 hmac 码发送给易宝
2) 易宝会连接银行，进行支付
3) 支付后，浏览器会以重定向方式访问 网站回调程序，易宝在收到银行转账后，会以 Socket 方式通知网站
* 在浏览器重定向通知后，提供给用户支付成功显示页面 （不要去修改订单状态）
* 在网站收到易宝点对点通知后，回复易宝 success，修改订单状态

数字签名原理：将数据 使用密钥和算法加密后 获得数字签名 hmac，将数据和数字签名 hmac 一起发送给易宝，易宝采用同样密钥和算法对数据进行加密获得数字签名 hmac，比较请求中数字签名 hmac 与加密后获得数字签名 hmac 是否一致，如果一致 签名 hmac 有效（数据没有被篡改 ）

流程入口： 未支付订单可以进行支付，管理员不允许支付

orders.jsp 点击在线支付
1) pay.jsp 银行选择页面
2) PayServlet 准备易宝支付参数
* 提供密钥和算法 商家编号 真实的 将 PaymentUtil 复制 utils 包 ， 将 merchantInfo.properties 复制 src
* responseURL 支付成功后 回调地址 ，该地址会收到 浏览器重定向和服务器点对点两次通知 ，确保该地址 可以被易宝访问 (estore 项目必须要配置缺省虚拟主机)
将 PayServlet 准备易宝参数，传递 JSP 确认
3) confirm.jsp 通过 form 的隐藏域 提交参数给易宝网址 https://www.yeepay.com/app-merchant-proxy/node

4) CallbackServlet 回调程序
收到浏览器重定向和服务器点对点两个通知
浏览器重定向，显示支付成功，不要修改订单状态
服务器点对点，修改订单状态，回复 success

将支付相关页面进行权限控制
* 千万不要将/callback 配置 user.txt 中，因为/callback 是由易宝访问，不应该有权限控制

销售榜单导出功能

获得商品销售情况，需要查询 orderitem 表 ------- 统计已支付订单项内容

1) 榜单中存在哪些信息？（已支付订单中商品）
商品信息 products 表
销售数量 orderitem 表
订单支付情况 orders 表

select * from products,orderitem,orders where products.id = orderitem.product_id and orderitem.order_id = orders.id ;

进行商品分组查询 group by

select products.* , sum(orderitem.buynum) totalSaleNum from products,orderitem,orders where products.id = orderitem.product_id and orderitem.order_id = orders.id and orders.paystate = 1 group by products.id order by totalSaleNum desc;

2) 榜单文件是什么格式？
导出 Excel 使用 POI 类库

csv 格式文件 ，逗号分隔文件

1) 信息当中有,在两端加 双引号
2) 信息当中有" 在之前加双引号 转义

文件下载
设置 Content-Type、Content-Disposition 头信息
文件流输出 （输出文件内容）

Excel 默认读取字符集 gbk

作业：
作业功能：编写定时器，在 ServletContextListener 中启动，每晚 12 点启动，将当天的未支付订单删除

第二十一天 框架学习之 java 基础加强

注解

注解作用：取代传统配置文件，通知编译器、虚拟机 该 Java 程序如何运行 。
* 区分注解 和 注释 区别？ 注释 给程序员阅读代码说明 ，注解给 虚拟机读取 关于类如何运行信息

典型应用：使用注解取代配置文件，采用反射技术读取注解的信息

JDK 自带三个注解的使用

ava5.0 程序允许使用注解技术 ------- @interface
JDK 中自带三个 最基本注解
@Override ： 表示该方法是对父类方法覆盖 ------ 编译时检查是否构成覆盖 （从 JDK6.0 开始，可以用于方法实现）
@Deprecated ：用于方法过时声明 ----- 考虑原来 API 被新的 API 取代，或者某些方法存在安全问题，声明该方法已经过时
@SuppressWarnings ： 用于在类编译时，忽略警告信息

自定义注解与应用

注解的应用编程
1、注解类定义
2、在目标类应用注解 ---- 重点
3、通过反射技术，获取注解信息，控制目标类如何运行

1、注解类定义
使用@interface 定义
属性语法： 返回值 名称() default 默认值 ;

注意：
属性的类型 String、基本数据类型、枚举、Class 、其它注解类型、以上数据类型相应一维数组
特殊属性 value, 如果一个注解只有一个属性 value ，在使用该注解时，省略 value=

元注解：注解的注解，在自定义注解时，应用这些元注解来修饰 自定义注解类
@Retention : 指定注解信息在哪个阶段存在 Source Class Runtime
@Target : 指定注解修饰目标对象类型 TYPE 类、接口 FIELD 成员变量 METHOD 方法

@Documented ：使用该元注解修饰，该注解的信息可以生成到 javadoc 文档中
@Inherited ：如果一个注解使用该元注解修改，应用注解目标类的子类都会自动继承该注解

***** @Retention @Target 是自定义注解必须使用两个元注解

示例 1 银行最大转帐金额

1) 使用 properties 配置最大转账金额
2) 使用注解取代配置文件
1) 定义银行信息注解 BankInfo

```
@Retention(RetentionPolicy.RUNTIME)
// 运行时使用
@Target(ElementType.METHOD)
public @interface BankInfo {
    // 最大转账金额
    double maxTransferMoney();
}
```

2) 在目标程序中应用注解

```
@BankInfo(maxTransferMoney = 50000)
public void transfer(String inAccount, String outAccount, double money) {
...
}
```

3) 在程序运行时，通过反射技术获取注解的信息
java.lang.reflect.AnnotatedElement 提供获取注解的方法
getAnnotation(Class<T> annotationClass) 获取指定注解的信息
isAnnotationPresent(Class<? extends Annotation> annotationClass) 判断该对象上是否存在目标注解

反射 Class、Constructor、Field、Method 都实现了该接口

步骤一：获得注解修饰目标对象对应反射对象 （Class、Constructor、Field、Method ）
步骤二：判断目标对象上是否存在该注解 isAnnotationPresent(Class<? extends Annotation> annotationClass)
步骤三：获得该注解信息 getAnnotation(Class<T> annotationClass)

配置文件 配置信息很灵活进行修改，为什么还需要注解技术呢？
最早：没有配置文件，数据在程序代码中，不方便修改
后来：通过配置文件，将动态数据提取到配置文件中，方便修改
再后来：随着系统业务越来越复杂，配置文件大小越来越大，使用配置文件最大缺点代码可读性变差、配置文件过大，维护及其不便 ，注解的出现简化配置文件维护不便、可读性差问题。

示例 2 JDBC 连接参数注解

```
create table products(
    id int primary key auto_increment,
    name varchar(40),
    price double
);

insert into products values(null,'冰箱',3000);
insert into products values(null,'洗衣机',2000);
insert into products values(null,'电视',5000);
```

将数据库驱动导入工程

注解定义
在目标类或者方法上应用注解
通过反射在运行时读取注解信息

Servlet3.0 特性

注解配置

注解技术产生 JDK5.0 例如：@override ， 在实际开发中注解经常用来取代配置文件

```
<servlet>
    <servlet-name>
    <serlvet-class>
```

```
</servlet>

<servlet-mapping>
    <servlet-name>
    <url-pattern>
</servlet-mapping>
```

使用@WebServlet 取代 ： @WebServlet(value="/hello")
@WebFilter 取代过滤器配置 ： @WebFilter(value="/hello")
@WebListener 取代监听器配置 ： @WebListener

@WebInitParam 为 Servlet 和 Filter 传递初始化参数

***** 注解不能取代 web.xml 所有功能 ， 必须配置 welcome-file 、error-page
****** metadata-complete web-app 元素的属性
设置为 true 将不支持注解技术
不设置 或者 设置为 false 将支持注解 技术

异步支持

步骤一 对 Servlet 或者 Filter 添加异步支持 asyncSupported=true
步骤二 开启异步线程

```
AsyncContext asyncContext = request.startAsync();
new Thread(new Executor(asyncContext)).start();
```

* 通过 AsyncContext 对象获得 request response servletcontext

通过

```
AsyncContext ctx = req.startAsync();
ctx.addListener(new AsyncListener() {
    public void onComplete(AsyncEvent asyncEvent) throws IOException {
        // 做一些清理工作或者其他
    }
    ...
});
```

文件上传支持

在 Servlet3.0 之前 编写文件上传 ：需要 jsp-smartupload 、commons-fileupload 第三方 jar 包
从 Servlet3.0 之后，Servlet 规程中内置 文件上传 API 支持

1、提供@MultipartConfig ，如果一个 Servlet 使用该注解，在 Servlet 中解析 request 请求时，

将根据 multipart/form-data 格式进行解析
* 默认 按照 urlencoded 格式解析

2、使用 request.getParameter 获得普通表单域内容
* 从 Servlet3.0 开始可以 使用 getParameter 获得上传表单中 普通字段值
* 乱码解除 request.setCharacterEncoding

3、获得文件上传域信息 使用 Part 类
request.setCharacterEncoding 解决上传文件名乱码问题

*** 注意 getParameter 使用 必须在 getPart 之前，文件乱码处理才会生效

动态代理

动态代理 可以拦截对真实业务对象的访问

注意：
1、代理对象需要和真实业务对象 提供相同行为方法
2、代理对象 拦截真实对象访问（阻止）、修改真实对象访问参数、修改真实对象操作返回值

代理原理：程序中获得真实业务对象，通过真实业务对象生成代理对象，通过代理对象间接访问真实业务对象
* 代理对象 阻止访问、修改参数和返回值

开发步骤

1、使用 Java 中动态代理，必须存在接口
Proxy 类，通过真实业务对象 ClassLoader 和 真实业务对象 实现 Interface ，在内存中虚拟一个代理类对象

2、使用 Proxy 类，根据真实业务对象 生成代理对象
newProxyInstance(ClassLoader loader, Class<?>[] interfaces, InvocationHandler h)

3、实现 invoke 方法
不改变原来代码逻辑 ： return method.invoke(liudehua, args); // liudehua 真实业务对象
阻止目标访问、修改参数和返回值
```
if(method.getName().equals("xxx 方法")){
  // 再对 xxx 方法 进行增强 改动
}
```

***** invoke 方法中会操作真实业务对象 ， 通常不会使用 proxy 对象

动态代理、包装、继承 都是 Java 中 方法增强的手段 ---------- 动态代理和包装 灵活性要比继承 更好

示例 1

web 工程中请求参数乱码处理

示例 2

基于动态代理和注解 实现权限管理 系统模型
1、系统存在若干功能， 将功能需要权限 定义注解
2、在业务层操作类 上面添加这些注解信息
3、web 层调用业务层，通过动态代理技术，获得注解信息 ，判断用户是否具有相应权限

```
create table users(
   id int primary key auto_increment,
   username varchar(40),
   password varchar(40)
);

insert into users values(null,'aaa','111');
insert into users values(null,'bbb','111');
insert into users values(null,'ccc','111');



create table privileges(
    id int primary key auto_increment,
    name varchar(40)
);

insert into privileges values(null,'添加图书');
insert into privileges values(null,'修改图书');
insert into privileges values(null,'查看图书');
insert into privileges values(null,'删除图书');

多对多表关系
create table userprivilege(
    user_id int ,
    privilege_id int,
    foreign key(user_id) references users(id),
    foreign key(privilege_id) references privileges(id),
```

```
    primary key(user_id,privilege_id)
);

insert into userprivilege values(1,1);
```

book.jsp 点击图书四个功能 --- 调用 BookService 中相应四个方法执行

4) 完成登录功能 ---- 将登陆信息保存 session
5) 将权限信息 定义为注解 对象，修饰 BookService
6) 提供工厂类， 将接口和实现类 进行解耦合 ---- 在 Servlet 不需要知道实现类具体实现

BookService bookService = new BookServiceImpl(); ========= > BookService bookService = BookFactory.getBookService();

7) 动态代理中，根据用户查询具有所有权限
select privileges.name from privileges,userprivilege where privileges.id = userprivilege.privilege_id and userprivilege.user_id = ?;

类加载器

类加载器（也是 java 程序），负责读取.class 字节码文件，将文件内容加载到内存中，生成 Class 类对象
Hello.java ---- Hello.class ------ Class 对象 Hello

JVM 类加载器初始化层次结构
1、BootStrap 引导类加载器 ----- JRE/lib/rt.jar 及其它该目录 jar 包
2、ExtClassLoader 扩展类加载器 ---- JRE/lib/ext/*.jar
3、AppClassLoader 应用类加载器 ---- 加载当前工程 classpath 下 jar 包 WEB-INF/lib 、WEB-INF/classes

4、自定义类加载器 ，加载特殊指定目录类

* 所有类加载器都是 java.lang.ClassLoader 子类 （BootStrap 加载器除外 ，该加载器是用 C 语言实现的）

类加载器： 全盘负责委托机制
1、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。（防止一个类被多个加载器加载）
2、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

实验：
javax.activation.MimeType 存在 rt.jar --------- BootStrap

demo4 编译没错，运行有错

demo5
java.lang.ClassCastException: javax.activation.MimeType cannot be cast to javax.activation.MimeType
* 两个不同加载器 加载同一个类之后，会生成两个 Class 类对象，不能转换

结论：
1、JVM 中加载器采用全盘负责委托机制
2、同一个类 被不同加载器加载后，会生成不同 Class 对象
3、了解 自定义类加载器