



# Java程序设计

2019 春季

彭启民

[pengqm@aliyun.com](mailto:pengqm@aliyun.com)


# JavaBean

- JavaBean 是可重用的Java组件，使用 `java.beans`包开发的，是 Java 2 标准版的一部分。必须可以在设计环境(design environment)中运行。在设计环境中允许终端用户制定其外观和行为。
  - 可实例化的`public`类，具有无参数的构造函数
  - 符合一致性设计模式
    - `set`和`get`方法访问成员属性，所有的属性私有
    - 其他Java 类可以通过自省机制(反射机制)发现和操作JavaBean 的属性

- **JavaBean**有3个接口面，可以独立进行开发。
  - **JavaBean**可以调用的方法。
  - **JavaBean**提供的可读写的属性。
  - **JavaBean**向外部发送的或从外部接收的事件。

JavaBean 通常用作 GUI 窗口小部件

JavaEE中的Enterprise Bean： 分布式商业对象。

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

## ■ JavaBean可分为两种

- 有用户界面（UI，User Interface），必须继承 `java.awt.Component`, 用于交互，如按钮、滚动条、数据库视图
- 无用户界面，主要负责处理事务（如数据运算，操纵数据库）
  - 仍然可以使用应用程序构造器可视化地进行组合。
  - JSP访问的通常为无用户界面的JavaBean。

## ■ Javabean 的使用方式

- 当作普通的类嵌入 java 代码
- 在 jsp 中 页面处理时使用 jsp 标记符方式

<!-- 使用JAVABEAN id代表实例化对象的名称 -->

```
<jsp:useBean id="person" scope="page" class="pojo.Person"></jsp:useBean>
```

<!--name与jsp:useBean中声明的Id -->

```
<jsp:setProperty name="person" property="*"></jsp:setProperty>
```

```
<h3><%=person.getName() %></h3>
```

```
<h3><%=person.getAge() %></h3>
```

<!-- 可以使用param来指定参数内容 -->

```
<jsp:setProperty name="person" property="name" param="age"/>
```

```
<jsp:setProperty name="person" property="age" param="name"/>
```

```
<h3><%=person.getName() %></h3>
```

```
<h3><%=person.getAge() %></h3>
```



# Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Hello extends HttpServlet {
    protected void service(HttpServletRequest request,
        HttpServletResponse response)
        throws ServletException, IOException {
        //ServletOutputStream out = response.getOutputStream();
        PrintWriter out = response.getWriter();
        out.println("Hello world!");
    }
}
```

# JSP简介

- **JSP (Java Server Pages)** 是一种动态网页技术标准，在传统的网页HTML文件 (\*.htm, \*.html) 中加入Java程序片段 (**Scriptlet**) 和 **JSP** 标记 (**tag**)，就构成了JSP网页 (\*.jsp)
- <http://www.javasoft.com/products/jsp>。





## Apache Tomcat Versions

Apache Tomcat<sup>®</sup> is an open source software implementation of the Java Servlet and JavaServer Pages technologies. Different versions of Apache Tomcat are available for different versions of the Servlet and JSP specifications. The mapping between the specifications and the respective Apache Tomcat versions is:

Servlet Spec	JSP Spec	EL Spec	WebSocket Spec	JASPIC Spec	Apache Tomcat version	Actual release revision	Supported Java Versions
4.0	TBD (2.4?)	TBD (3.1?)	TBD (1.2?)	1.1	9.0.x	9.0.0.M10 (alpha)	8 and later
3.1	2.3	3.0	1.1	1.1	8.5.x	8.5.5	7 and later
3.1	2.3	3.0	1.1	N/A	8.0.x (superseded)	8.0.37 (superseded)	7 and later
3.0	2.2	2.2	1.1	N/A	7.0.x	7.0.70	6 and later (7 and later for WebSocket)
2.5	2.1	2.1	N/A	N/A	6.0.x	6.0.45	5 and later
2.4	2.0	N/A	N/A	N/A	5.5.x (archived)	5.5.36 (archived)	1.4 and later
2.3	1.2	N/A	N/A	N/A	4.1.x (archived)	4.1.40 (archived)	1.3 and later
2.2	1.1	N/A	N/A	N/A	3.3.x (archived)	3.3.2 (archived)	1.1 and later

# Servlet

JSP页面转译成Servlet时，主要包含三个部分：

```
public void _jspInit(){ ..}
```

// JSP网页最先执行此方法，初始化

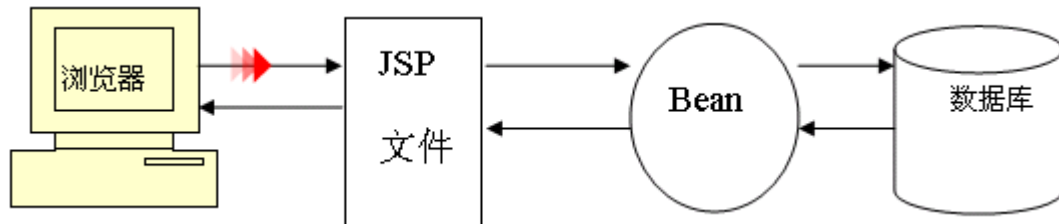
```
public void _jspDestory(){...} //JSP网页最后执行的方法
```

```
public void _jspService(HttpServletRequest request,  
    HttpServletResponse response)
```

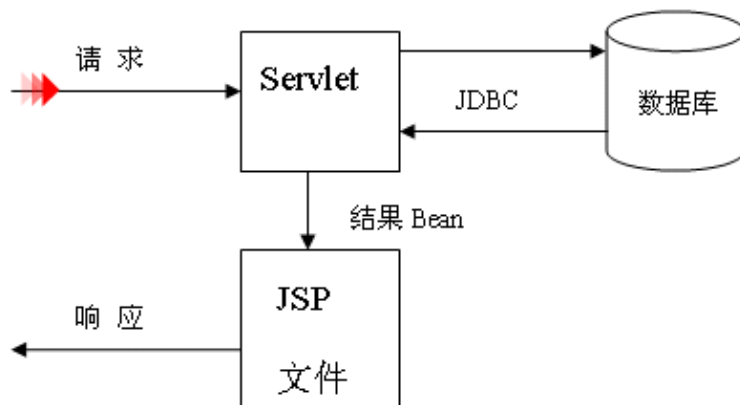
```
    throws java.io.IOException, ServletException {
```

## JSP的两种访问格式

(1) 请求一个**JSP文件**：**JSP文件**将访问**Bean**或其他能生成动态内容并发送到浏览器的组件。



(2) 请求一个**Servlet**：客户端直接请求一个**Servlet**来生成动态内容，并调用**JSP文件**将内容发送到浏览器。



## ■ JSP运行环境的配置

- 如果想用**JSP**来开发网站，那么需要安装一些应用程序服务器软件，提供对**JSP/Servlet**的支持。这样就可使用主服务器响应普通网页的请求，应用程序服务器软件响应动态网页的请求。

# 相关工具下载

- 为了实验JSP技术，首先需要建立运行环境
  - 下载安装好JDK
  - 下载安装一个支持JSP的服务器
    - 在<http://tomcat.apache.org/>处下载Tomcat
      - JSWDK: JavaServer Web Development Kit。

- 服务器参数保存在...\webserver.xml中。
- 文档目录在缺省状态下为...\webpages
- 主文档在缺省状态下为index.html和index.jsp。也就是说访问<http://localhost:8080>等于访问...\webpages\index.html。

# JSP设置

- 在Web服务的文档目录此目录下建立子目录，例如...\\webpages\\test，浏览器中对应的访问地址为<http://localhost/test>，为了使得这个子目录能执行JSP程序，必须：
  - 在webserver.xml中的 <Service> </Service>节加入：

```
<WebApplication id="test" mapping="/test" docBase="webpages/test" />
```
  - 建立WEB-INF子目录 (...\\webpages\\test\\WEB-INF)，并从....\\webpages\\WEB-INF目录中复制过来以下四个文件：
    - mappings.properties
    - mime.properties
    - servlets.properties
    - webapp.properties。

## ■ 启动JSP服务器（命令行）

- startserver

- 为避免与可能已有的Web服务器（例如IIS、PWS等）冲突，在webserver.xml中指定使用8080端口。

- 浏览器访问地址：<http://localhost:8080>或者<http://127.0.0.1:8080>

- 看到欢迎页说明JSP实验环境已经建成。

## ■ 关闭Web服务器

- stopserver



# JSP网页示例

- 创建文本文件hi.jsp，保存在...\webpages目录下，其内容如下：

```
<html>
<head>
<title>Hi-JSP实验</title>
</head>
<body>
<%
    String Msg = "This is a JSP test.";
    out.print("Hello World!");
%>
<h2><%=Msg%></h2>
</body>
</html>
```

- Web服务器会执行JSP 文件中用 `<%以及%>`括起来的Java程序语句

- `out.print`是将文字输出到网页

- 语句 `<%= 变量 | 表达式%>`的作用是将Java Scriptlet中变量或表达式的值输出到网页.

- `System.out.println`输出在服务器日志中。

- 访问地址

- <http://localhost:8080/hi.jsp>,

# JSP表格



```
<TABLE BORDER=2>
```

```
<%
```

```
for ( int i = 0; i < n; i++ ) {
```

```
%>
```

```
<TR>
```

```
<TD>Number</TD>
```

```
<TD><%= i+1 %></TD>
```

```
</TR>
```

```
<%
```

```
}
```

```
%>
```

```
</TABLE>
```

# 显示时间

## ■ Jsp显示当前时间

```
<%@ page import="java.util.*, java.text.*" %>
<HTML>
<HEAD>
<TITLE>JSP to display the current time</TITLE>
</HEAD>
<BODY>
The current time is:
<%
Date now = new Date();
out.println(DateFormat.getInstance().format(now));
%>
</BODY>
</HTML>
```

# 文件包含

- 用可视化HTML编辑器，例如FrontPage、Dreamweave等设计网站的框架结构

- ```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>我的主页 </title>
</head>
<body>
<table border="0" width="100%" cellpadding="4" cellspacing="0" align="center">
<tr>
<td width="100%" colspan="2" bgcolor="#837ED1" align="center"><font face="隶书" color="#FFFF00" size=5>主页标题</font>
</td>
</tr>
<tr>
<td bgcolor="#837ED1" width="15%" valign="top" align="center"><br>
<font color="#FFFFFF">选项</font><p><font color="#FFFFFF">选项</font></p>
<p><font color="#FFFFFF">选项</font></p>
<p><font color="#FFFFFF">.....</font></p>
<p> </p>
</td>
<td width="85%" valign="top">
```
- -----以上保存为 top.htm-----以下保存为 bot.htm-----

```
</td>
</tr>
</table>
</body>
</html>
```

- 每个JSP文件都有如下结构:

```
<%@ include file="top.htm" %>
<%
// 实现某些功能
%>
```

```
<%@ include file="bot.htm" %>
```

维护网站的界面也相对比较容易，只要修改top.htm和bot.htm，就能影响到所有网页。

- Mypage.jsp


```
<%@ include file="top.htm" %>
<%
String Msg = "This JSP test.";
out.print("Hello World!");
%>
<h2><%=Msg%></h2>
<%@ include file="bot.htm" %>
```

- 在浏览器的地址栏中键入<http://localhost:8080/mypage.jsp>.  
这样网站的界面就能统一起来，而设计者可以集中精力在功能模块上处理用户登录、连接数据库、发送email等等。


# JSP访问数据库

- 建立数据库及表格
- 插入记录
- 例：Access

lyb: 表		
	字段名称	数据类型
id		自动编号
	name	文本
	mail	文本
	web	文本
	qq	文本

- 
- 建立数据源
  - “控制面板” → “管理工具” → “数据源(ODBC)”
  - 选择 “系统DNS”选项卡，单击添加
  - 选择 “Dirver do Microsoft Access (\*.mdb)”



- 
- 注意！一定要用系统数据源，不能用用户数据源
    - 所有的jsp、servlet程序，需要连接jdbc-odbc驱动的话，都只能用系统数据源，不能用用户数据源

## 创建新数据源



选择您想为其安装数据源的驱动程序 (S)。

- | 名称                                            |
|-----------------------------------------------|
| Driver da Microsoft para arquivos texto (*.*) |
| <b>Driver do Microsoft Access (*.mdb)</b>     |
| Driver do Microsoft dBase (*.dbf)             |
| Driver do Microsoft Excel (*.xls)             |
| Driver do Microsoft Paradox (*.db )           |
| Driver para o Microsoft Visual FoxPro         |
| Microsoft Access Driver (*.mdb)               |
| Microsoft Access-Treiber (*.mdb)              |
| Microsoft dBase Driver (*.dbf)                |
| Microsoft dBase VFP Driver (*.dbf)            |
| Microsoft Excel Driver (*.xls)                |
| Microsoft Paradox Driver (*.db)               |
| Microsoft Visual FoxPro Driver                |

< 上一步 (B)

完成

取消

Macromedia Dreamweaver 绿色最简优化版 - [无标题文档 (Untitled-1\*)]

文件(F) 编辑(E) 查看(V) 插入(I) 修改(M) 文本(T) 命令(C) 站点(S) 窗口(W) 帮助(H)

常用

Untitled-1\*

代码 拆分 设计 标题: 无标题文档

```
10
11 <div align="center" class="style1">JSP连接Access数据库</div>
12 <br>
13 <hr>
14 <p><%
15 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //载入驱动程序类别
16 Connection con = DriverManager.getConnection("jdbc:odbc:jspdata"); //建立数据库链接,jspdata为ODBC数据源
17 //建立Statement对象
18 Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
19 ResultSet.CONCUR_READ_ONLY);
20 ResultSet rs = stmt.executeQuery("select * from lyb"); //建立ResultSet(结果集)对象,并执行SQL语句
21 %>
22 //</p>
```

NUMB1数据表中记录如下

编号	姓名	E-mail	网站	QQ
JSP	JSP	JSP	JSP	JSP

如果您能看到表格中的数据,说明连接数据库成功!

<body><p>

781 x 210 3 K / 1 秒

属性

JSP 编辑

框架

(不包含框架)

文件

代码

代码片断 参考

书籍:

主题:

设计

应用程序

数据库 绑定 服务器行为 组件

+ - 文档类型: JSP

+ mydb

标签检查器

```
<%@ page contentType="text/html; charset=gb2312" language="java" import="java.sql.*"
errorPage="" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>JSP连接数据库</title>
<style type="text/css">
<!--
.style1 {
font-size: 20px;
font-weight: bold;
}
-->
</style>
</head><body>
<div align="center" class="style1">JSP连接Access数据库</div>
<br>
<hr>
<p><%
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //载入驱动程序类别
Connection con = DriverManager.getConnection("jdbc:odbc:jspdata"); //建立数据库链接
,jspdata为ODBC数据源名称
//建立Statement对象
Statement stmt = con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
ResultSet rs = stmt.executeQuery("select * from testTable"); //建立ResultSet(结果集)对象
, 并执行SQL语句
%>
```

```
□ </p>
<p align="center">NUMB1数据表中记录如下</p>
<table width="640" border="1" align="center" bordercolor="#7188e0">
<tr bgcolor="d1d1ff">
<th width="49">编号</th>
<th width="90">姓名</th>
<th width="126">E-mail</th>
<th width="221">网站</th>
<th width="80">QQ</th>
</tr>
<%
while(rs.next())
{
%>
<tr bgcolor="#f8f8f8">
<th><%= rs.getString(1) %></th>
<th><%= rs.getString(2) %></th>
<th><%= rs.getString(3) %></th>
<th bgcolor="#f6f6f8"><%= rs.getString(4) %></th>
<th><%= rs.getString(5) %></th>
</tr>
<%
}
rs.close();
stmt.close();
con.close();
%>
</table>
<p align="center"><br>
如果您能看到表格中的数据，说明连接数据库成功！</p>
</body>
</html>
```



# Java的存储

- 寄存器：最快 JVM
- 堆栈：对象引用
- 堆：存所有JAVA对象
- 静态存储：存放程序运行时一直存在的数据
- 常量存储：在程序代码内部 `public static final`
- 非RAM存储

`String str = new String("test");` //创建了几个对象?  
//1个或2个

- 堆是一个运行时数据区，储存着正在运行的应用程序通过new、newarray、anewarray和multianewarray等指令建立的所有对象，但不需要显式地释放,由垃圾回收来负责
- 垃圾回收是一种动态存储管理技术，它自动地释放不再被程序引用的对象，按照特定的垃圾收集算法来实现资源自动回收的功能。
  - 非JVM规范，但是由于内存的有限性，JVM在实现的时候都有一个由垃圾回收所管理的堆。



# 永远不要考虑对象清理


- 垃圾回收器
- 作用域清晰


# 垃圾回收器


- **System.gc():** 用于强制进行终结动作
  - 回收空间
  - 使堆中的对象紧凑排列
- **常见工作方式**
  - 引用计数
  - 停止—复制
  - 标记—清扫

# 清理

- 对象可能不被垃圾回收
- 垃圾回收并不等于“析构”
  - 通常在析构方法中进行释放对象占用的相关资源的操作。
- 垃圾回收只与内存有关

- 
- **gc**即垃圾收集机制是指jvm用于释放那些不再使用的对象所占用的内存。
  - **java**语言并不要求jvm有**gc**，也没有规定**gc**如何工作。不过常用的jvm都有**gc**，而且大多数**gc**都使用类似的算法管理内存和执行收集操作。

- 
- 垃圾收集的目的是清除不再使用的对象。
  - **gc**通过确定对象是否被活动对象引用来确定是否收集该对象。

- 
- 垃圾收集的一个潜在的缺点是它的开销影响程序性能。
    - **Java**虚拟机必须追踪运行程序中有用的对象，而且最终释放没用的对象。这一个过程需要花费处理器的时间。
    - 垃圾收集算法的不完备性，早先采用的某些垃圾收集算法就不能保证**100%**收集到所有的废弃内存。

- 命令行参数透视垃圾收集器的运行
  - 在命令行中有一个参数-verbosegc可以查看Java使用的堆内存的情况：

`java -verbosegc classfile`



■ 例子:

■ class TestGC

```
{  
    public static void main(String[] args)  
    {  
        new TestGC();  
        System.gc();  
        System.runFinalization();  
    }  
}
```



- 在这个例子中，一个新的对象被创建，由于它没有使用，所以该对象迅速地变为可回收
- 编译执行：

```
java -verbosegc TestGC
```

- 结果：

[Full GC 168K->97K(1984K), 0.0253873 secs]


- 箭头前后的数据**168K**和**97K**分别表示垃圾收集GC前后所有存活对象使用的内存容量，说明有**168K-97K=71K**的对象容量被回收，括号内的数据**1984K**为堆内存的总容量，收集所需要的时间是**0.0253873秒**（这个时间在每次执行的时候会有所不同）。

## ■ **finalize**方法透视垃圾收集器的运行

- 在JVM垃圾收集器收集一个对象之前，一般要求程序调用适当的方法释放资源，但在没有明确释放资源的情况下，Java提供了缺省机制来终止化该对象心释放资源，这个方法就是**finalize（）**。

**protected void finalize() throws Throwable**

- 在**finalize()**方法返回之后，对象消失，垃圾收集开始执行。原型中的**throws Throwable**表示它可以抛出任何类型的异常。
- 按创建对象相反的顺序释放：子类的**finalize()**方法须首先调用基类的**finalize()**方法！

- 
- 当垃圾回收器将要释放无用对象的内存时，先调用该对象的**finalize**（）方法。
  - 如果在程序终止前垃圾回收器始终没有执行垃圾回收操作，那么垃圾回收器将始终不会调用无用对象的**finalize**（）方法。

# Java垃圾回收的特点

## ■ 垃圾收集发生的不可预知性

- 由于实现了不同的垃圾收集算法和采用了不同的收集机制，所以它有可能是定时发生，有可能是当出现系统空闲CPU资源时发生，也有可能是和原始的垃圾收集一样，等到内存消耗出现极限时发生，这与垃圾收集器的选择和具体的设置都有关系。

- 使用System.gc()可以不管JVM使用的是哪一种垃圾回收的算法，都可以请求Java的垃圾回收。

- 当你调用**System.gc()**时，JVM不会马上去执行！

- Java中提供了一些和垃圾收集打交道的类，而且提供了一种强行执行垃圾收集的方法--调用**System.gc()**，但这同样是个不确定的方法。Java 中并不保证每次调用该方法就一定能够启动垃圾收集，它只不过会向JVM发出这样一个申请，到底是否真正执行垃圾收集，一切都是个未知数。

- 现在有许多种不同的垃圾收集器，垃圾收集的实现和具体的JVM 以及JVM的内存模型有非常紧密的关系。
  - 当垃圾收集开始时停止应用程序的运行
  - 当垃圾收集开始时也允许应用程序的线程运行
  - 在同一时间垃圾收集多线程运行

## ■ 垃圾收集的精确性

- 垃圾收集器能够精确标记活着的对象
- 垃圾收集器能够精确地定位对象之间的引用关系。
- 前者是完全地回收所有废弃对象的前提，否则就可能造成内存泄漏。而后者则是实现归并和复制等算法的必要条件。所有不可达对象都能够可靠地得到回收，所有对象都能够重新分配，允许对象的复制和对象内存的缩并，这样就有效地防止内存的支离破碎。

■ 例：

```
static String str = new String("a string object");
```

这里str是String对象(“a string object”)的一个引用。

```
令 str= new String("another string object");
```

这时候String对象(“a string object”)失去引用，所以该对象将会被释放，但str被关联到新的String，所以str本身不会被释放。

```
当 final static String str = new String("a string object");
```

这个str是final，所以其指到的对象永远不会失去引用，所以永远不会被释放。



## ■ 建议

- 良好的编程习惯和严谨的编程态度永远是最重要的。
- 尽早释放无用对象的引用。
  - 大多数程序员在使用临时变量的时候，都是让引用变量在退出活动域(scope)后，自动设置为null，暗示垃圾收集器来收集该对象，还必须注意该引用的对象是否被监听，如果有，则要去掉监听器，然后再赋空值。
- 有时为确保对象资源的明确释放，可以编写自己的finalize方法。



# 泛型

# Java的泛型(generics,模板类)

## ■ Java的模板类

- `public void write(Integer i, Integer[] ia);`  
`public void write(Double d, Double[] da);`

的范型版本为

`public <T> void write(T t, T[] ta);`

# Java的泛型(generics,模板类)

## ■ Java的模板类

- 在定义带类型参数的类时，在紧跟类命之后的<>内,指定一个或多个类型参数的名字，同时也可以对类型参数的取值范围进行限定，多个类型参数之间用,号分隔。
- 定义完类型参数后，可以在定义位置之后的类的几乎任意地方（静态块，静态属性，静态方法除外）使用类型参数，
- 就像使用普通的类型一样。

# Java的泛型(generics,模板类)

## ■ Java的模板类

- Java泛型是JDK 5中引入的一个新特性，允许在定义类和接口的时候使用类型参数（**type parameter**）。
- 泛型本质上是提供类型的"类型参数"，它们也被称为参数化类型（**parameterized type**）或参量多态（**parametric polymorphism**）。
- 模板类的模板参数只能是参数类型，成员变量类型等，模板名是确定的。
- 声明的类型参数在使用时用具体的类型来替换。
- 运行期，模板参数会被当作**Object**来处理
- 使用模板类的类型安全，只是利用编译器的类型检查，来自动保证运行期的类型强转的正确与安全。

# Java的泛型(generics,模板类)

- 定义完类型参数后，可以在定义位置之后的方法的任意地方使用类型参数，就像使用普通的类型一样。例如：

```
public <T, S extends T> T testGenericMethodDefine(T  
t, S s){  
    ...  
}
```

- 注意，父类定义的类型参数不能被子类继承。
- 当对类或方法的类型参数进行赋值时，要求对所有的类型参数进行赋值。否则，将得到一个编译错误。

# Java的泛型(generics,模板类)

- Java的模板类

甚至可以对类型参数使用通配符!

```
List<?> unknownList;
```

```
List<? extends Number> unknownNumberList;
```

```
List<? super Integer> unknownBaseLineIntgerList;
```



# Java的泛型(generics,模板类)

- 对带类型参数的类进行类型参数赋值有两种方式

- 声明类变量或者实例化时

```
List<String> list;
```

```
list = new ArrayList<String>;
```

- 继承类或者实现接口时

```
public class MyList<E> extends ArrayList<E>  
implements List<E> {...}
```



- 当调用泛型方法时，编译器自动对类型参数进行赋值，当不能成功赋值时报编译错误。例如

```
public <T> T testGenericMethodDefine3(T t, List<T> list){  
    ...  
}  
public <T> T testGenericMethodDefine4(List<T> list1, List<T> list2){  
    ...  
}
```

```
Number n = null;  
Integer i = null;  
Object o = null;  
testGenericMethodDefine(n, i); //此时T为Number, S为Integer  
testGenericMethodDefine(o, i); //T为Object, S为Integer
```

```
List<Number> list1 = null;  
testGenericMethodDefine3(i, list1) //此时T为Number
```

```
List<Integer> list2 = null;  
testGenericMethodDefine4(list1, list2) //编译报错
```

# Java的泛型(模板类、泛类型)

## ■ Java的模板类

- Java的泛型就是创建一个用类型作为参数的类。目前类型参数只接受引用类型，不接受基本类型。

List<Integer> ✓      List<int> ✗ (以后可能支持?)

# Java的泛型(模板类、泛类型)

- 注意!
- 在泛型中List<Object>, List<String>是两种不同的类型, 之间没有继承关系, 即使String继承了Object。下面的代码是错误的!

```
List<String> ls = new ArrayList<String>();
```

```
List<Object> lo = ls;
```



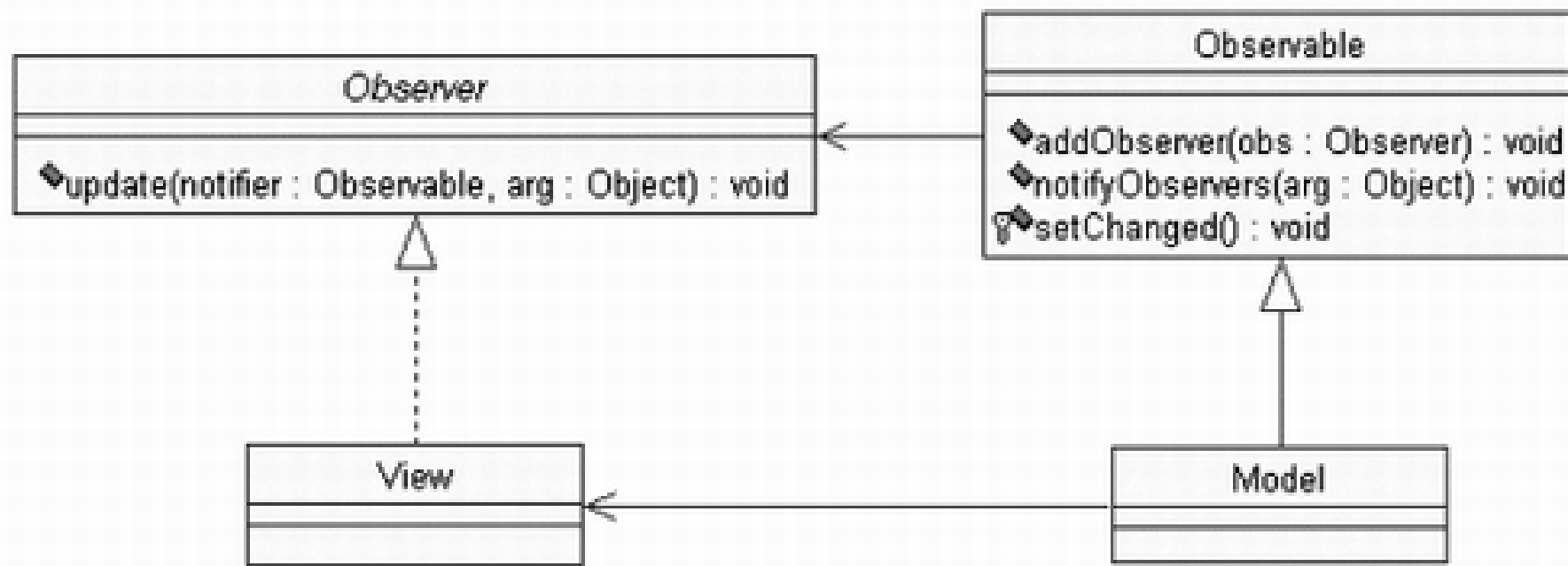
# 设计模式与框架

# MVC模式介绍

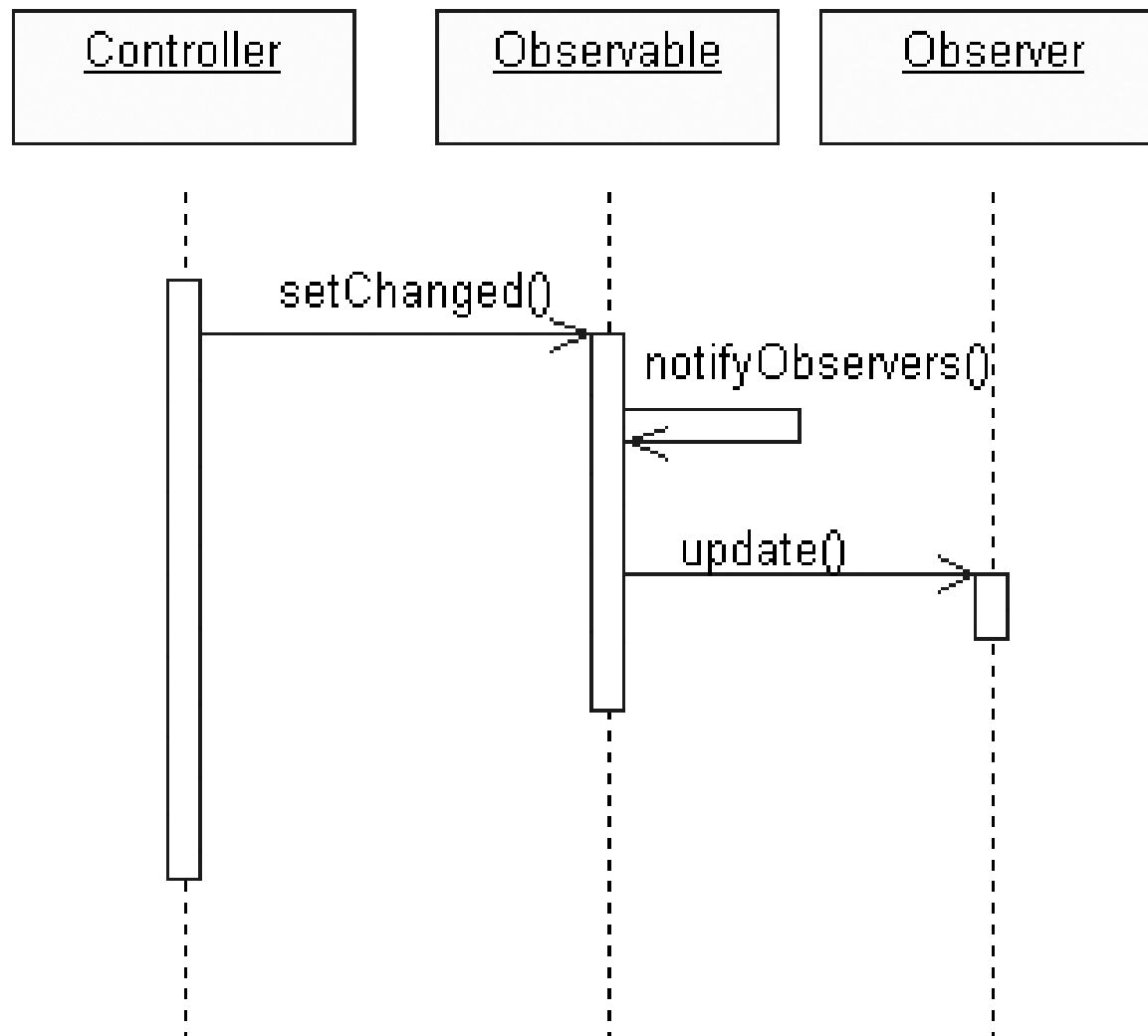
- 模型—视图—控制器(Model-View-Controller, MVC)模式是为那些需要为同样的数据提供多个视图的应用程序而设计的。它很好地实现了数据层与表示层的分离，特别适用于开发与用户图形界面有关的应用程序。
- 模式中基本结构定义为：
  - 控制器 用来处理用户命令以及程序事件的；
  - 模型 维护数据并提供数据访问方法；
  - 视图 数据的显示。

# 基本MVC模式

- Java通过专门的类Observable及Observer接口来实现MVC设计模式。



# MVC模式的时序图



# 一个例子

- 当用户在图形化用户界面输入一个球体的半径时，显示该球体的体积与表面积。该程序主要由三个类构成。
  - Sphere类扮演Model的角色
  - TextView类为View角色
  - SphereWindow类为Controller角色



- **Model类Sphere**，必须扩展**Observable**类，因为在**Observable**类中，方法**addObserver()**将视图与模型相关联，当模型状态改变时，**Sphere**通过方法**notifyObservers()**通知视图。

- **View**类的角色**TextView**类必须实现接口**Observer**，需实现其中的方法**update()**。有了这个方法，当模型**Sphere**类的状态发生改变时，与模型相关联的视图中的**update()**方法就会自动被调用，从而实现视图的自动刷新。


- 
- SphereWindow类作为Controller，它主要新建Model与View，将View与Model相关联，并处理事件

```
import java.util.Observer;
import java.util.Observable;
public class TextView extends JPanel implements Observer{
    .....
    public void update(Observable o, Object arg){
        Sphere balloon = (Sphere)o;
        radiusIn.setText(" "+f3.format(balloon.getRadius()));
        volumeOut.setText(" "+f3.format(balloon.volume()));
        surfAreaOut.setText(" "+f3.format(balloon.surfaceArea()));
    }
    .....
}

import java.util.Observable;
class Sphere extends Observable{
    .....
    public void setRadius(double r){
        myRadius = r;
        setChanged();
        notifyObservers();
    }
    .....
}
```

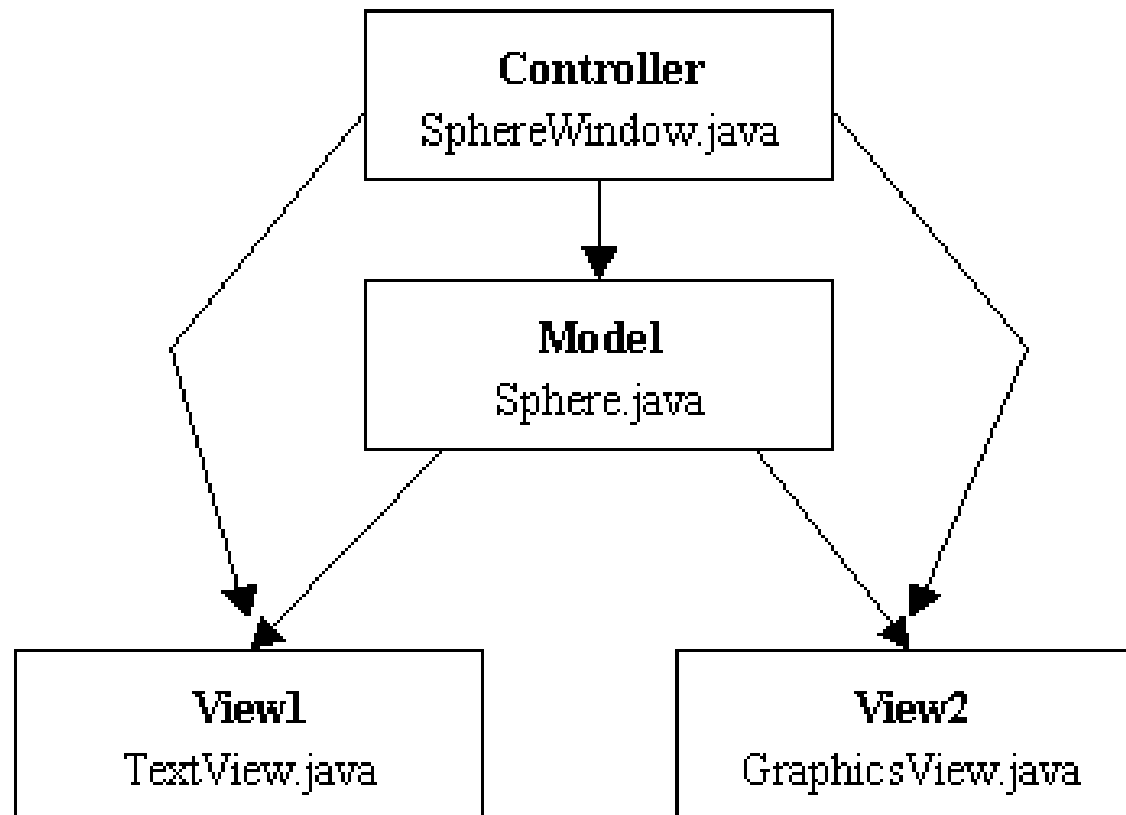
```
import javax.swing.JPanel;
import javax.swing.JFrame;
import java.awt.Container;
import javax.swing.JTextField;
class SphereWindow extends JFrame{
    public SphereWindow(){
        super("Spheres: volume and surface area");
        model = new Sphere(0, 0, 100);
        TextView view = new TextView();
        model.addObserver(view);
        view.update(model, null);
        view.addActionListener(this);
        Container c = getContentPane();
        c.add(view);
    }

    public void actionPerformed(ActionEvent e){
        JTextField t = (JTextField)e.getSource();
        double r = Double.parseDouble(t.getText());
        model.setRadius(r);
    }
}
```


- 
- 该程序通过Java中的MVC模式编写的，具有良好的可扩展性
    - 实现一个模型的多个视图
    - 采用多个控制器
    - 当模型改变时，所有视图将自动刷新
    - 所有的控制器将相互独立工作

# 一个模型、两个视图和一个控制器

- 当用户在图形化用户界面输入一个球体的半径，程序除显示该球体的体积与表面积外，还将图形化显示该球体。
  - Model类及View1类不需改变
  - Controller中的SphereWindows类增加另一个视图，并与Model发生关联即可。







```
public SphereWindow(){
    super("Spheres: volume and surface area");
    model = new Sphere(0,0,100);
    TextView tView = new TextView();
    model.addObserver(tView);
    tView.addActionListener(this);
    tView.update(model, null);
    GraphicsView gView = new GraphicsView();
    model.addObserver(gView);
    gView.update(model, null);
    Container c = getContentPane();
    c.setLayout(new GridLayout(1, 2));
    c.add(tView);
    c.add(gView);
}
```

# MVC的优点

- 多个视图对应一个模型，可满足用户需求的变化，实现多种方式访问应用
- 模型返回的数据不带任何显示格式，因而这些模型也可直接应用于接口的使用。
- 应用被分离为三层，灵活性增强。
  - 有时改变其中的一层就能满足应用的改变，如一个应用的业务流程或者业务规则的改变只需改动MVC的模型层。

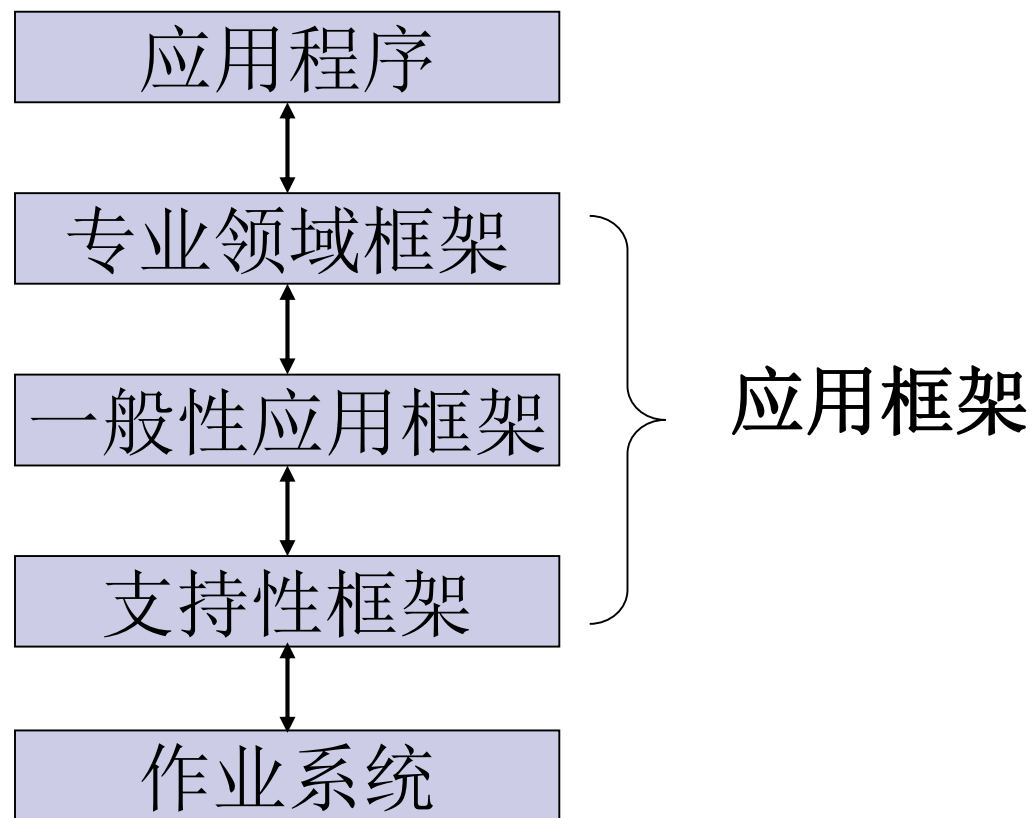
# MVC的不足

- 增加了系统结构和实现的复杂性。
- 视图与控制器间的过于紧密的连接。妨碍了他们的独立重用。
- 视图对模型数据的低效率访问。

# 框架Framework

- 框架，其实就是某种应用的半成品，就是一组组件，供你选用完成你自己的系统。简单说就是使用别人搭好的舞台，你来做表演。而且，框架一般是成熟的，不断升级的软件。
- 框架一般处在低层应用平台（如**JavaEE**）和高层业务逻辑之间的中间层。

# Framework



# 优秀框架所必须具备的

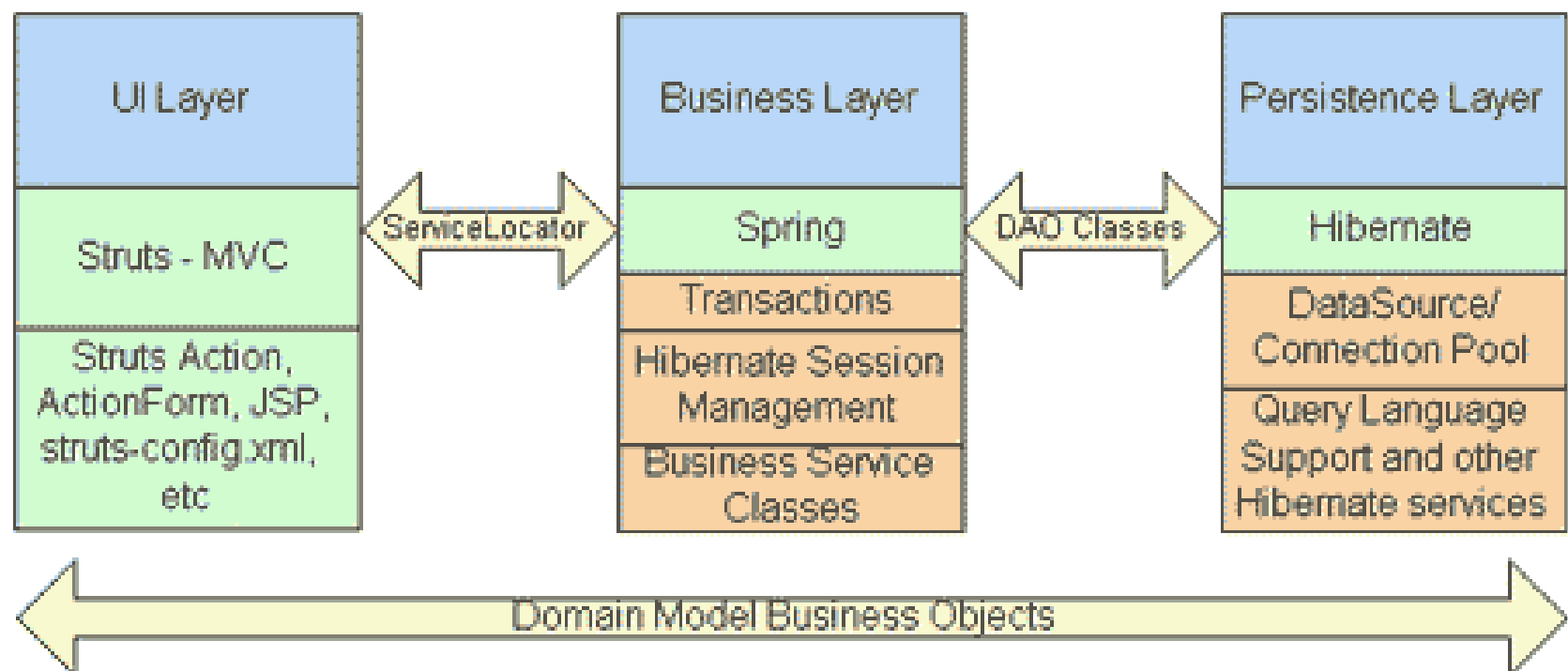
- **开放性：**一个框架不能对开发者限制太多，也不能将开发者封装在一个黑盒子中，应该可以让开发者越过该框架操作该框架力不能及的功能。
- **良好性能：**框架应该是轻量且运行时不是主要占据CPU，该框架要提供缓存或池功能提升性能，同时框架本身也必须提高性能，不能因为采取新技术而忽略性能要求，这是很多号称新技术革命的框架缺乏的。
- **状态管理：**状态是业务逻辑中主要操作对象，是等同于数据库的重要概念，一个框架应该提供领域模型的Session状态管理。

# JavaEE Patterns


## ■ 特点

- 与平台、语言相关，描述的是JavaEE平台上利用Java如何解决设计问题
- 所解决问题的规模较大，在JavaEE框架的基础上解决构件技术的选择、构件之间的协作等问题
- 模式抽象层次较高，同时提供了若干具体实现的细节，称为策略。
- 注重性能的优化

# SSH





- 
- **Struts**主要用于显示层和控制层
    - Struts2 Struts2.1.....
  - **Spring**用于业务逻辑层用的事物管理和对象管理
    - Spring2.0 Spring2.5 Spring3.0....
  - **Hibernate**用于持久层进行数据库操作
    - Hibernate 2 Hibernate3.0 ....

# 设计模式vs.框架

## ■ 不同的研究领域

- 设计模式是软件的知识体，设计模式研究的是一个设计问题的解决方法，提升框架的设计水平
- 框架是软件，是一个应用体系结构，是一种或多种设计模式和代码的混合体

## ■ 统一的思想

- 设计可以被重用

## ■ 设计模式比框架更抽象

- 设计模式在每一次被复用时，都需要被实现
- 框架不仅能被学习，还能被直接执行和复用

## ■ 设计模式是比框架更小的体系结构元素


- 一个典型的框架往往包含了多个设计模式，反之不然

## ■ 框架比设计模式更加特例化

- 框架总是针对一个特定的应用领域
- 框架是OO系统获得最大复用的方式，较大的OO应用将会由多层彼此合作的框架组成



# 其他话题



性能优化  
高性能计算  
大数据  
云计算  
人工智能



感谢同学们的支持！  
祝同学们进步！