第2章 软件体系结构

2.2 软件体系结构核心概念

刘其成

计算机与控制工程学院

ytliuqc@163.com

2018-9

主要内容

- **2.2.1 SA的核心概念**
- **2.2.2** 组件
- **2.2.3** 连接件

2.2.1 SA的核心概念

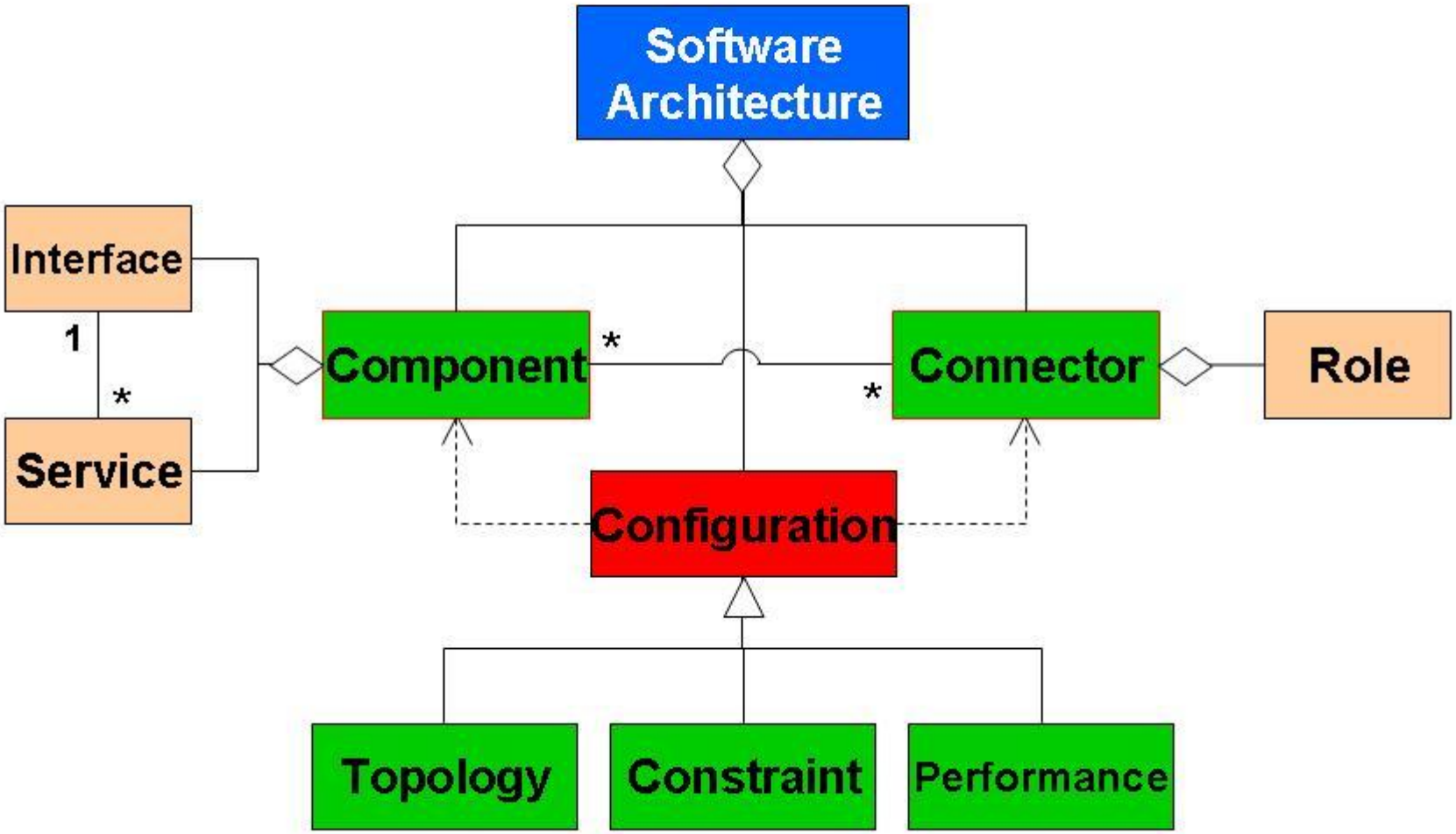
回忆：SA的定义

- 软件体系结构(SA)：
  - 提供了对软件结构、行为和属性的高级抽象
  - 从一个较高的层次来考虑组成系统的组件、组件之间的连接，以及由组件与组件交互形成的拓扑结构
  - 这些要素应该满足一定的限制，遵循一定的设计规则，能够在一定的环境下进行演化。
  - 反映系统开发中具有重要影响的设计决策，便于各种人员的交流，反映多种关注，据此开发的系统能完成系统既定的功能和性能需求。

体系结构 = 组件 + 连接件 + 拓扑结构 + 约束 + 质量

Architecture = Components + Connectors + Topology + Constraints + Performance

SA中的核心概念


Software Architecture
Interface
1
*
Service
Component
*
Connector
*
Role
Configuration
Topology
Constraint
Performance

2.2.2 组件(Component)

组件(Component)

- 生活中，组装一台电脑时，人们可以选择多个组件，例如内存、显卡、硬盘等，
- 一个组装电脑的人不必关心内存是怎么研制的。不同的电脑可以安装相同的内存，内存的功能完全相同，但它们是在不同的电脑里面，一台电脑的内存发生了故障并不影响其它的电脑；
- 也可能两台电脑连接了一个共享的组件：集线器，如果集线器发生了故障，两台电脑都受到同样的影响——无法连接网络。

组件(Component)

- 广义上讲，组件是具有某种功能的可复用的软件结构单元，是为组装服务的，是组成软件系统的计算单元或数据存储单元。
- 例如：
  - 共享变量
  - 函数、子程序
  - 对象、类
  - 文件
  - 程序
  - 数据库
  - ....

复用：

代码复用、
设计复用、
分析复用、
测试复用...

函数

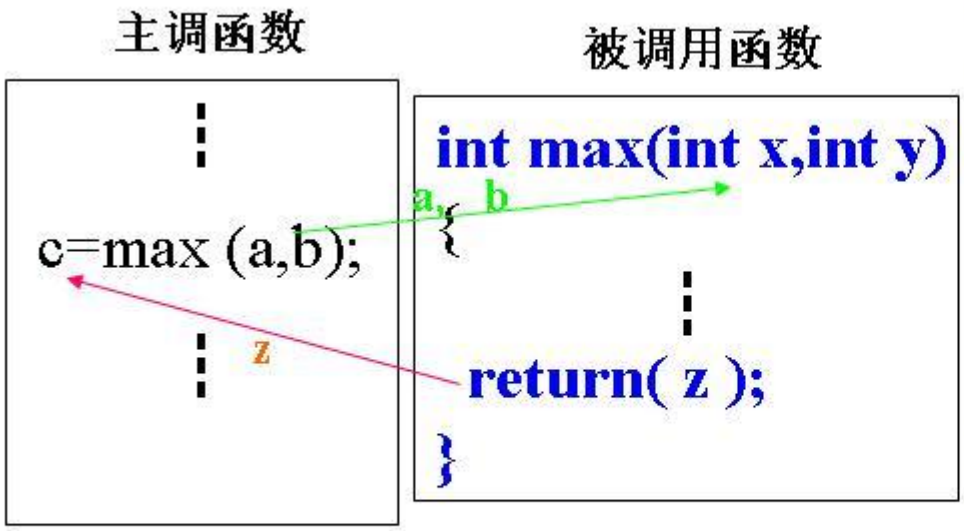
- 一个源程序文件，由一个或多个函数以及其他有关内容组成，是一个编译单位，函数不是一个编译单位。
- C程序的执行总是从main函数开始，调用其它函数后回到main函数，在main函数中结束整个程序的运行；
- 所有的子函数都是平行的，任何子函数都不属于其他函数；

例

```
#include <stdio.h>
void printstar()
{ printf("**************\n");
}
void printmessage()
{printf("  Hello,world.\n");
 printstar();
}
void main()
{
    printstar();
    printmessage();
}
```

函数

一个C程序由若干个函数组成，各函数调用时经常要传递一些数据，调用函数把数据传递给被调用函数，经被调用函数使用后，一般会返回一个确定结果，在返回调用函数时，把这些结果带回调用函数。



例

```
#include <stdio.h>
int max(int x,int y)
{  int z;
   z = x > y ? x : y;
   return( z );
}
void main()
{ int a,b,c ;
   scanf("%d,%d",&a,&b);
   c=max(a,b);
   printf("The max is %d", );
}
```

各函数的信息往来主要是由参数传递和返回语句实现的

对象的访问

- 例 对象用作方法的参数

```
class Spot {
    private int x, y;
    Spot (int u, int v) {
        setX(u);  setY(v);
    }
    void setX(int x1) { x=x1; }
    void setY(int y1) { y=y1; }
    int getX() { return x; }
    int getY() { return y; }
}
class Trans {
    void move(Spot p, int h, int k) {
        p.setX(p.getX() + h);
        p.setY(p.getY() + k);
    }
}
```

```
class Test {
   public static void main(String args[]) {
      Spot s = new Spot(2, 3);
      System.out.println("s点的坐标:"
                + s.getX()+","+s.getY());
      Trans ts = new Trans();
      ts.move(s, 4, 5);
      System.out.println("s点的坐标:"
                +s.getX()+","+s.getY());
   }
}
```

```
D:\java Test
s点的坐标:2,3
s点的坐标:6,8
```

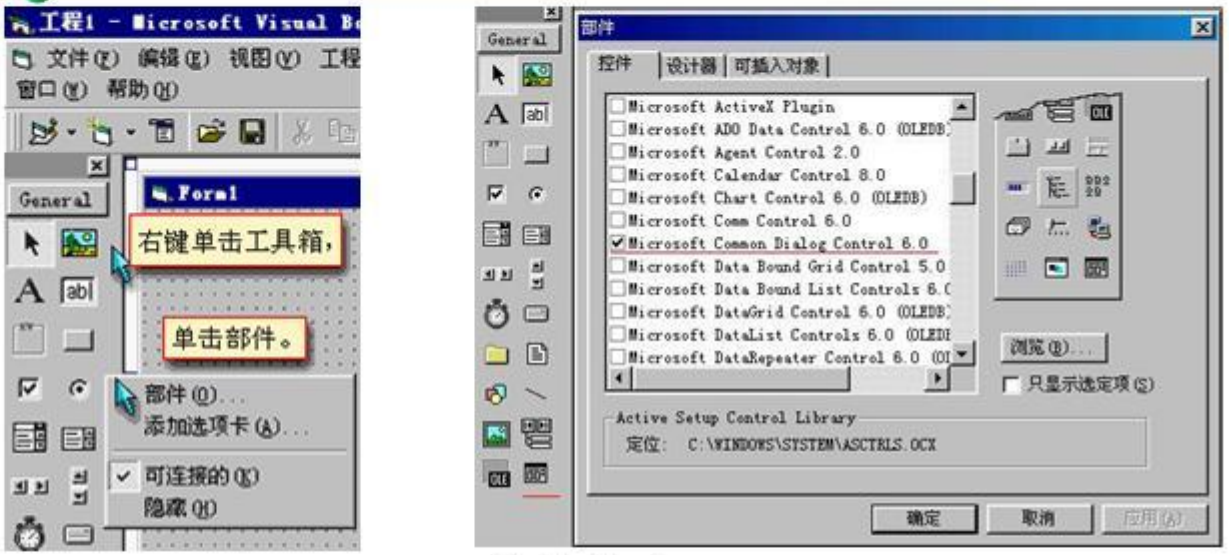
组件

可视化的编程环境中的组件

在可视化的编程环境中编程序，

可以把一个按钮组件或文本框组件拖放到应用程序窗体中，

还可以很方便地重新更改它的名字等属性。

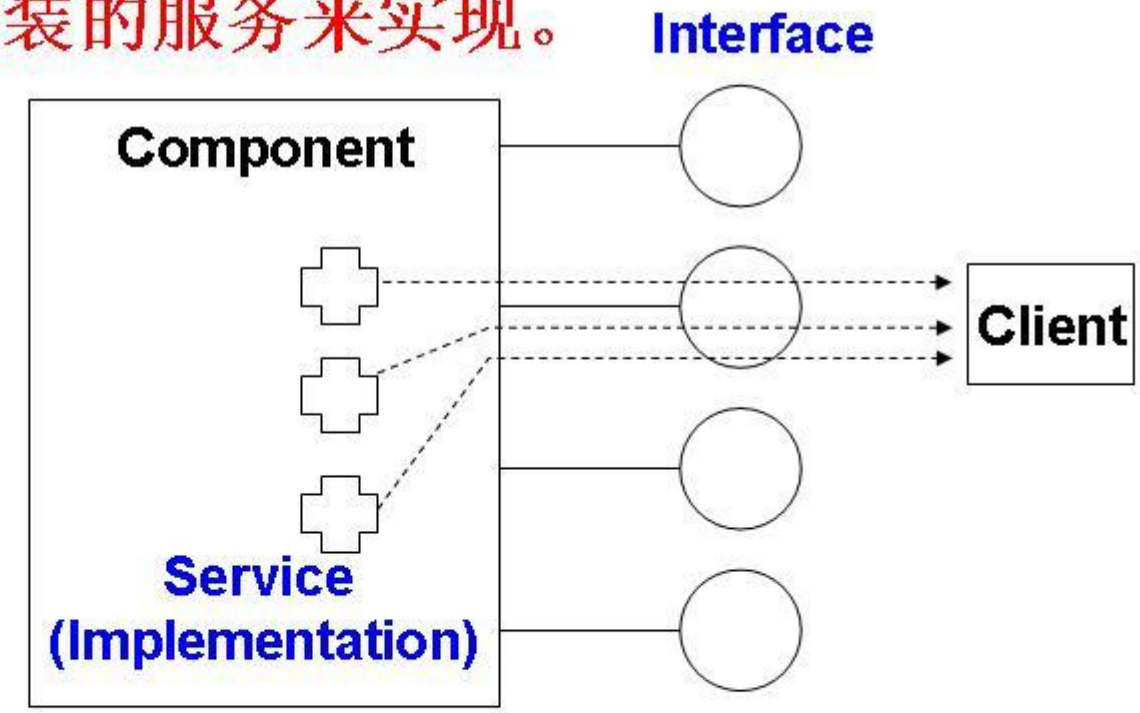


Visual Basic控件技术

PowerBuilder Datawindow组件技术

组件(Component)

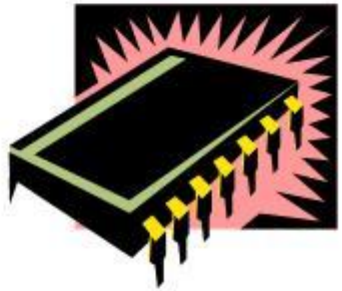
- 严格意义上讲，组件是一种可部署单元，它具有规范的接口规约和显式的语境依赖，而接口功能由组件内部封装的服务来实现。



- 由组件组装出的软件称为组件化软件。

接口(Interface)与服务(Service)

- 组件作为一个封装的实体，只能通过其接口(Interface)与外部环境交互，表示了组件和外部环境的交互点；
- 内部结构则被隐藏起来(Black-box)；
- 一个组件至少有一个接口；
- 一个组件可以提供多重接口；
- 组件接口与其内部实现应严格分开.



- 组件内部所实现的功能以服务(Service)的形式体现出来，并通过接口向外发布，进而产生与其它组件之间的关联。

- 组件的优点何在？快速开发、提高软件质量

组件的定义

- 一般认为，组件是指语义完整、语法正确和有可复用价值的单位软件，是软件复用过程中可以明确辩识的系统；结构上，它是语义描述、通讯接口和实现代码的复合体。
- 简单地说，组件是具有一定的功能，能够独立工作或能同其它组件装配起来协调工作的程序体，组件的使用同它的开发、生产无关。

粒度的定义(Granularity)

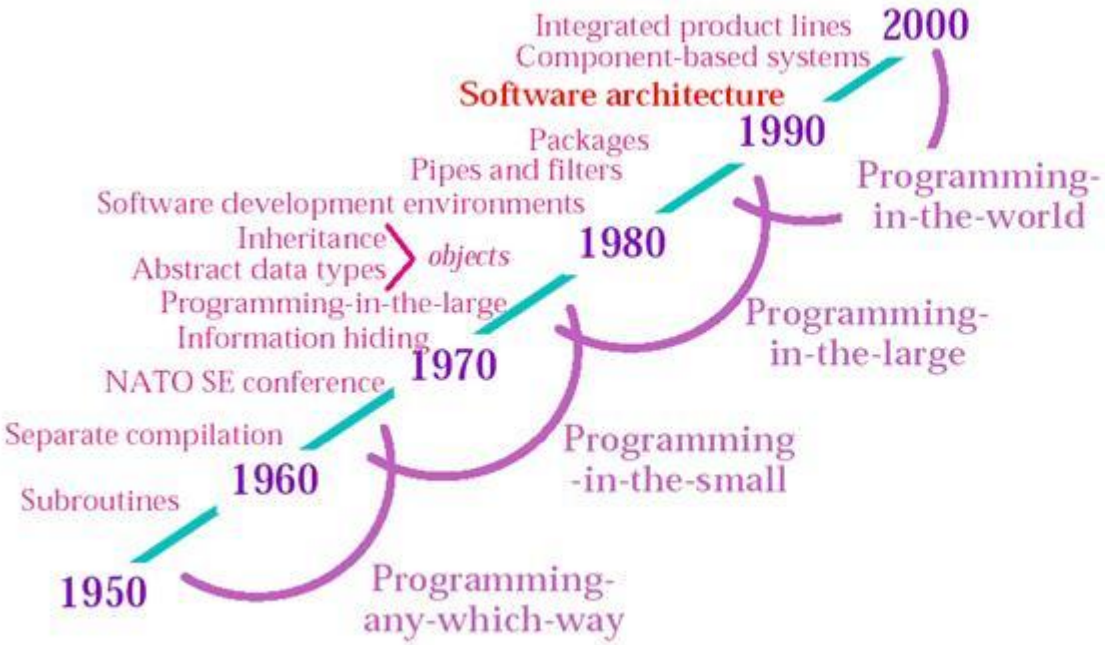
- 粒度：组件的相对大小、规模、细节程度或关注程度的一个属性
  - 国家-省份-地区-城市-街道
- 原子组件(Atomic component)：不可再分解。
  - 砖、瓦
  - 集成电路、芯片
  - 数据结构、函数
- 复合组件(composite component)：由其他原子组件与复合组件通过连接而成。
  - 预制板、房屋框架
  - 存储器、运算器、控制器
  - 对象、模块、子系统
- 组件的定义是递归的概念

粒度对组件的影响

- 恰当的粒度是很重要的
- 小粒度组件：灵活性高，复用度高，但交互复杂，使用效率低下
- 大粒度组件：正好相反。便于管理，但臃肿、庞大

体系结构中组件粒度的演变

- 系统 = 算法 + 数据结构 (1960's)
- 系统 = 子程序 + 子程序 (1970's)
- 系统 = 对象 + 对象 (1980's)
- 系统 = 组件 + 连接件 (1990's)
- 系统 = 服务 + 服务总线 (2000's)



组件的粒度

- 函数
  - 函数、子程序、方法
- 类
  - 类级的复用（代码复用）是以类为封装的单位，但这样的复用粒度还太小，不足以解决异构互操作和效率更高的复用。
- 类的组合
  - 通常讲的组件是对一组类的组合进行封装，并代表完成一个或多个功能的特定服务，也为用户提供了多个接口。整个组件隐藏了具体的实现，只用接口提供服务。

函数

- 一个源程序文件，由一个或多个函数以及其他有关内容组成，是一个编译单位，函数不是一个编译单位。
- C程序的执行总是从main函数开始，调用其它函数后回到main函数，在main函数中结束整个程序的运行；
- 所有的子函数都是平行的，任何子函数都不属于其他函数；

例

```
#include <stdio.h>
void printstar()
{ printf("**************\n");
}
void printmessage()
{printf("  Hello,world.\n");
 printstar();
}
void main()
{
    printstar();
    printmessage();
}
```

对象的访问

- 例 对象用作方法的参数

```
class Spot {
    private int x, y;
    Spot (int u, int v) {
        setX(u);  setY(v);
    }
    void setX(int x1) { x=x1; }
    void setY(int y1) { y=y1; }
    int getX() { return x; }
    int getY() { return y; }
}
class Trans {
    void move(Spot p, int h, int k) {
        p.setX(p.getX() + h);
        p.setY(p.getY() + k);
    }
}
```

```
class Test {
   public static void main(String args[]) {
      Spot s = new Spot(2, 3);
      System.out.println("s点的坐标:"
                + s.getX()+","+s.getY());
      Trans ts = new Trans();
      ts.move(s, 4, 5);
      System.out.println("s点的坐标:"
                +s.getX()+","+s.getY());
   }
}
```

```
D:\java Test
s点的坐标:2,3
s点的坐标:6,8
```

组件技术实例：J2EE中的组件技术

- JSP组件
- Servlet组件
- Javebean组件
- EJB组件
- 等

- J2EE中有3组9种组件：
- 客户端: 应用组件、JavaBean和Applet
- WEB服务器层：Servlet和Jsp
- 应用层服务器：四种EJB（有状态会话Bean、无状态会话Bean、实体和消息驱动Bean）

JavaBeans

- JavaBeans是一种规范，一种在Java（包括JSP）中使用可重复使用的Java组件的技术规范；按着Sun公司的定义，JavaBeans是一个可重复使用的软件组件。
- 同时，JavaBeans是一个Java的类，通过封装属性和方法具有某种功能或者处理某个业务，这样的Java类一般对应于一个独立的.java文件。
- 另外，当JavaBeans这样的一个Java类在具体的Java程序中被实例之后，有时也会将这样的一个JavaBeans的实例称为JavaBeans。

组件技术实例：JavaBean的实现

- JavaB
  codin
  - Impl
  - Prop
    - Pr
      pa
      na
    - Pr
  - Ever
    and
    the
    defi

```
package com.stardeveloper.bean.test;
public class SimpleBean implements java.io.Serializable {
        /* Properties */
        private String name = null;
        private int age = 0;
        /* Empty Constructor */
        public SimpleBean() {}
        /* Getter and Setter Methods */
        public String getName() {
                return name;
        }
        public void setName(String s) {
                name = s;
        }
        public int getAge() {
                return age;
        }
        public void setAge(int i) {
                age = i;
        }
}
```

e a
ty
ner)
ure to
s
ties.

- 如果类的成员变量的名字是**xxx**，那么为了更改或获取成员变量的值，即更改或获取属性，在类中就需要有两个方法。
  - **getXxx()**：用来获取属性**xxx**
  - **setXxx()**：用来修改属性**xxx**
  - 对于**boolean**类型的成员变量，即布尔逻辑类型的属性，允许使用“**is**”代替上面的“**get**”和“**set**”
- 类中的普通方法不适合上面的命名规则，但这个方法必须是**public**的。类中如果有构造方法，那么这个构造方法也是**public**的并且是无参数的。

组件技术实例：在JSP组件中使用JavaBean组件

```
<html>
<head>
        <title>SimpleBean Test Page</title>
</head>
<body>

<%-- Creating JavaBeans --%>
<jsp:useBean id="simple" class="com.stardeveloper.bean.test.SimpleBean">
        <jsp:setProperty name="simple" property="name" value="Faisal Khan" />
        <jsp:setProperty name="simple" property="age" value="24" />
</jsp:useBean>

<%-- Displaying JavaBean property's value --%>
<p>Name retrieved from JavaBean has the value of :
        <b><jsp:getProperty name="simple" property="name" /></b>.<br>
Age retrieved from JavaBean has the value of :
        <b><jsp:getProperty name="simple" property="age" /></b> years.<br>
</p>

</body>
</html>
```

- 为了在JSP页面中使用JavaBeans，必须使用JSP动作标签：useBean。useBean格式：
- <jsp:useBean id= "给JavaBeans起的名字" class= "创建JavaBeans的类" scope= "JavaBeans有效范围">
  </jsp:useBean>
- class=“com.stardeveloper.bean.test.SimpleBean”中包含包的名字，或者有如下的import指令：
  <@page import="com.stardeveloper.bean.test">

组件技术实例：JavaBean和JSP组件的部署

- JavaBean的部署
  - myWebsite/WEB-INF/classes/com/SimpleBean.class
- JSP的部署(不允许放在/WEB-INF/下)
  - myWebsite/testBean.jsp
- 其他语境依赖
  - 符合JSP规范应用服务器
  - JVM等

2.2.3 连接与连接件 (Connection and Connector)

连接(Connection)

- 连接(Connection)：组件间建立和维护行为关联与信息传递的途径；
- 连接需要两方面的支持：
  - 连接发生和维持的机制——实现连接的物质基础(连接的机制)；
  - 连接能够正确、无二义、无冲突进行的保证——连接正确有效的进行信息交换的规则(连接的协议)。
  - 简称“机制”(mechanism)和“协议”(protocol)。

连接的机制(Mechanism)

- 计算机硬件提供了一切连接的物理基础：
  - 中断、存储、堆栈、串行I/O、并行I/O、网络等；
- 连接实现机制：
  - 过程调用、回调、进程通讯、内存共享、同步/异步、远程过程调用、管道、消息传递、反射、动态链接、动态绑定、文件等；

连接的协议(Protocol)

- 协议(Protocol)是连接的规约(Specification)；
- 连接的规约是建立在物理层之上的有意义信息形式的表达规定
  - 对过程调用来说：参数的个数和类型、参数排列次序；
  - 对消息传送来说：消息的格式；
  - 对ODBC数据库连接来说：SQL语言；
  - 对Web Service连接而言：SOAP或REST协议；
- 目的：使双方能够互相理解对方所发来的信息的语义

连接的双方的角色

- 角色：连接的双方所处的不同地位的表达
  - 过程调用：调用方(Caller)和被调用方(Callee)
  - 管道：读取方(Reader)和写入方(Writer)
  - 消息传递：发送者(Sender)和接收者(Receiver)
  - …(不同的体系结构风格中拥有不同的角色)

连接件(Connector)

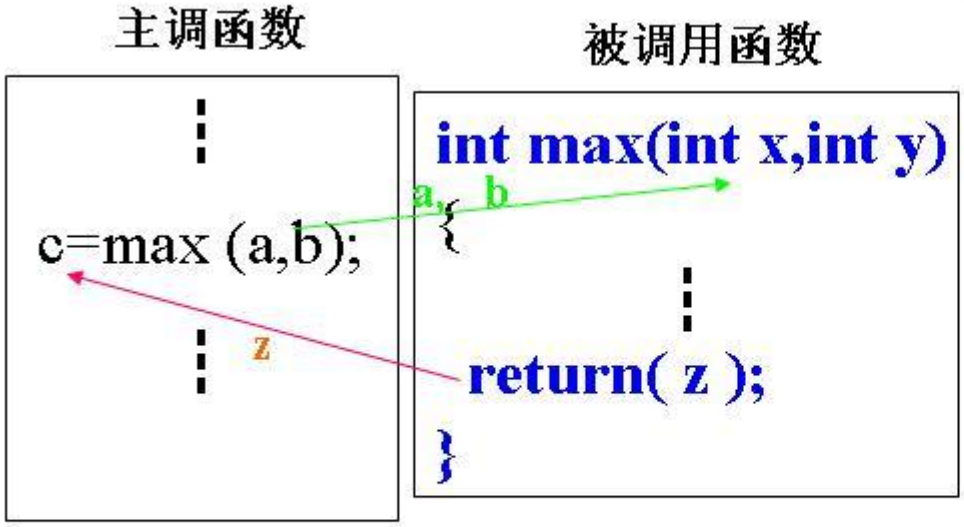
- 连接件(Connector)：表示组件之间的交互并实现组件之间的连接，连接件也可看作一类特殊的组件，区别在于：
  - 一般组件是软件功能设计和实现的承载体；
  - 连接件是负责完成组件之间信息交换和行为联系的专用组件。

连接件(Connector)

- 连接件有哪些？
  - 调用返回
  - 管道(pipe)
  - ……
  - 中间件(Middleware)
    - ODBC/JDBC
    - 应用服务器
    - WEB服务器
    - 消息中间件
    - .....

函数参数和函数的值——调用返回机制

一个C程序由若干个函数组成，各函数调用时经常要传递一些数据，调用函数把数据传递给被调用函数，经被调用函数使用后，一般会返回一个确定结果，在返回调用函数时，把这些结果带回调用函数。




```
例
#include <stdio.h>
int max(int x,int y)
{  int z;
   z = x > y ? x : y;
   return( z );
}
void main()
{ int a,b,c ;
   scanf("%d,%d",&a,&b);
   c=max(a,b);
   printf("The max is %d", );
}
```

各函数的信息往来主要是由参数传递和返回语句实现的

- 组件
  - 主程序**main()**函数
  - 子程序**max(a,b)**函数
- 连接件
  - **main()**函数中调用**max(a,b)**函数
  - **max()**函数将实参**a**、**b**分别传递给虚参**x**、**y**
  - 通过运算得到较大值**z**
  - 将**z**返回调用处，赋值给**main()**函数的变量**c**

函数的调用

函数调用的执行过程

1. 按从右到左的顺序，计算实参各表达式的值；
2. 按照位置，将实参的值一一传给形参；
3. 执行被调用函数；
4. 当遇到return(表达式)语句时，计算表达式的值，并返回主调函数。

例8-4 读程序，写出结果

```
#include <stdio.h>
                ❷
int  iabs( float x)
❸ { return (x>0 ? x : -x);
  }

void main( )
{   float  x=-1.2,y;
 ❹ y= iabs(2*x ) ❶
    printf( "x=%f , iabs(x)=%f\n",x,y);
}
```

对象的访问

- 例 对象用作方法的参数

```
class Spot {
    private int x, y;
    Spot (int u, int v) {
        setX(u);  setY(v);
    }
    void setX(int x1) { x=x1; }
    void setY(int y1) { y=y1; }
    int getX() { return x; }
    int getY() { return y; }
}
class Trans {
    void move(Spot p, int h, int k) {
        p.setX(p.getX() + h);
        p.setY(p.getY() + k);
    }
}
```

```
class Test {
   public static void main(String args[]) {
      Spot s = new Spot(2, 3);
      System.out.println("s点的坐标:"
                + s.getX()+","+s.getY());
      Trans ts = new Trans();
      ts.move(s, 4, 5);
      System.out.println("s点的坐标:"
                +s.getX()+","+s.getY());
   }
}
```

```
D:\java Test
s点的坐标:2,3
s点的坐标:6,8
```

- 组件
  - Spot、Trans、Test三个类
  - Spot类的对象s，Trans类的对象ts，Spot类的对象p
- 连接件
  - 在Test类里面创建Spot类的对象s、Trans类的对象ts，Trans类的move()方法的参数里面有Spot类的对象p
  - Test类使用Spot类的对象s，调用了Spot类的getX()和getY()方法
  - Test类使用Trans类的对象ts，调用了Trans类move()方法，并把实参Spot类的对象s传递给了虚参Spot类的对象p。

谢谢

2018年9月13日