软件不同设计的艺术
示例1：类的不同设计

战德臣

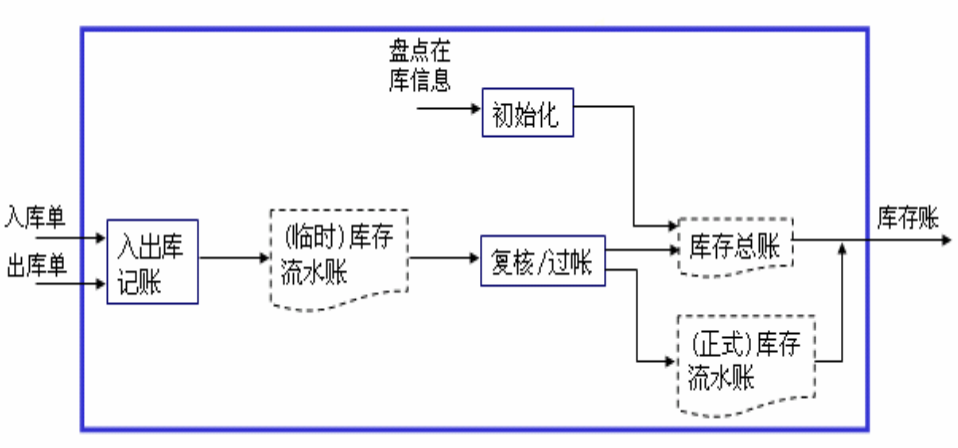
哈尔滨工业大学　教授.博士生导师

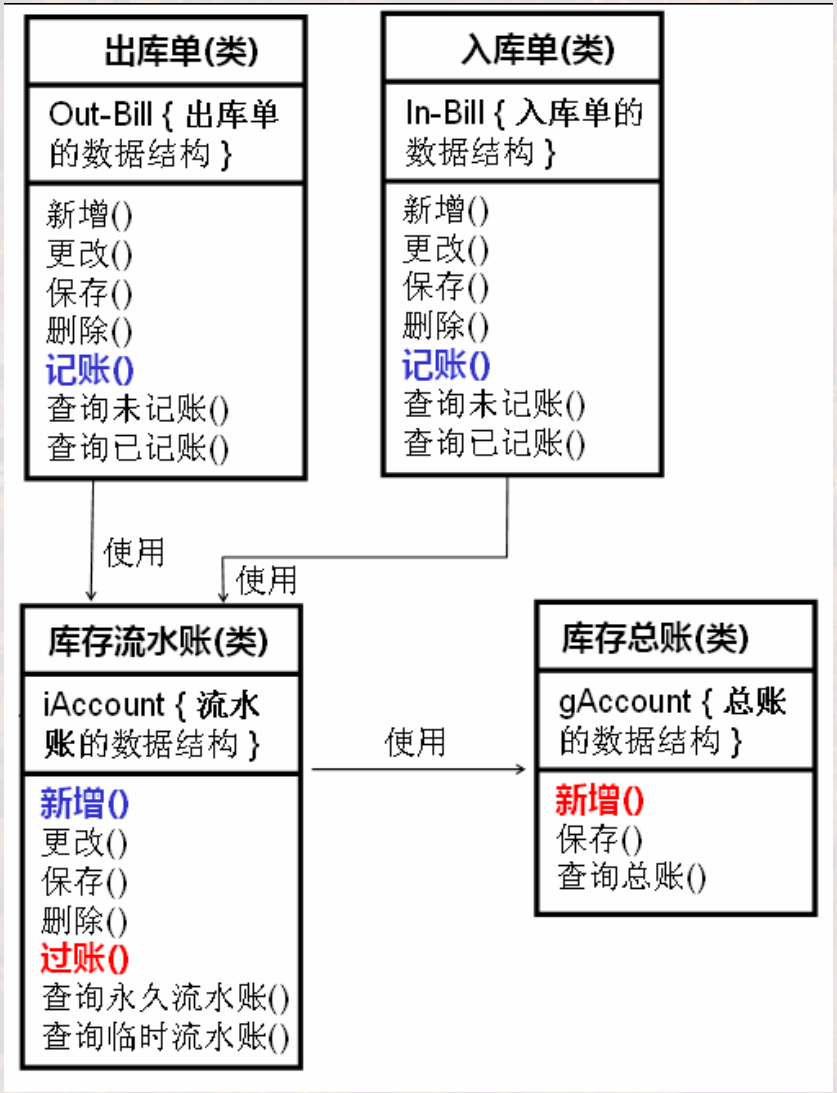
教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology

软件不同设计的艺术
(1)类的不同设计

战德臣 教授

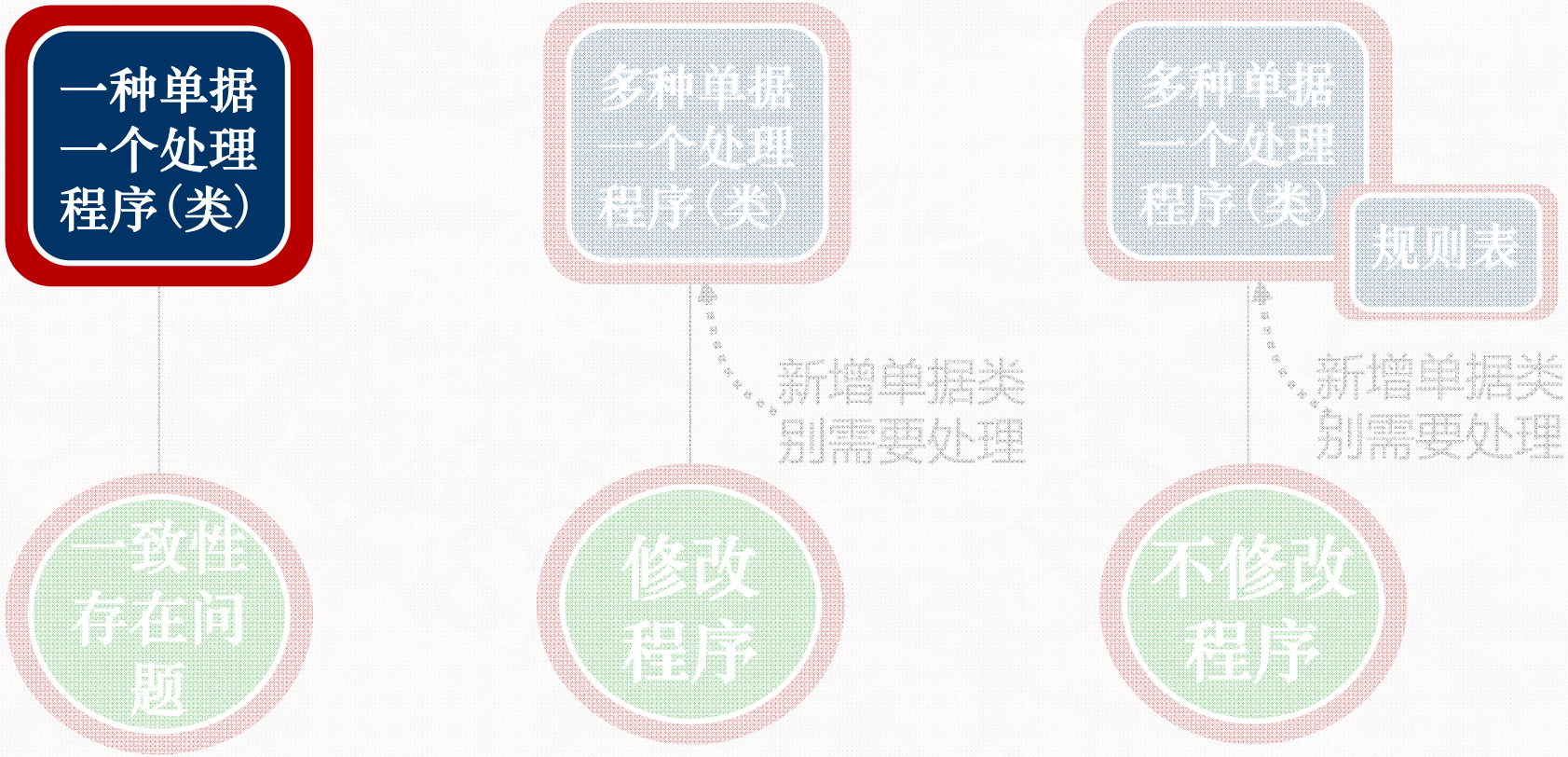




“类”应该如何设计呢?

战德臣 教授

库存账(类)的三种不同设计方案的对比



软件不同设计的艺术
(1)类的不同设计

出库单-记账-过账的紧耦合设计

Class 出库单-记账
{
Struct Out-Bill
{/* 定义存储**出库单_数据结构**的变量及其类型

出库单

出库单编号：_____ 仓库：_____
去向：_____ 用途：_____

物资名称	规格/型号	出库数量	计量单位	单价	总价	领料单号/提货单号	说明
MAT1		200	包	1000.00	200,000.00		
MAT2		100	个	150.00	15,000.00		

保管员：_____ 领料员：_____ 日期：_____

*/
}
Struct iAccount
{/* 定义存储**流水账_数据结构**的变量及其类型

库存流水账

日期	物资	入/出	数量	单价	金额
2012.06.01	M1	入	200	1,100.0000	220,000.00
2012.06.02	M1	出	80	1,100.0000	88,000.00
2012.06.03	M1	出	80	1,100.0000	88,000.00

*/
}
int Out-Func( )
{
//将 Out-Bill 中的数据，赋值给 iAccount 中的变量；
//将 iAccount 的数据写入临时流水账数据库中 Tmp-iAccount.
}
int OutTransfer()
{
//创建出库-过账对象。如 TransferObj = new 出库-过账；
//将待过账的流水账数据传递给出库-过账对象 TransferObj.
//调用 TransferObj.TranferFun()执行过账。
}
}

Class 出库-过账
{
Struct iAccount
{/* 定义存储**流水账_数据结构**的变量及其类型

库存流水账

日期	物资	入/出	数量	单价	金额
2012.06.01	M1	入	200	1,100.0000	220,000.00
2012.06.02	M1	出	80	1,100.0000	88,000.00
2012.06.03	M1	出	80	1,100.0000	88,000.00

*/
}
Struct gAccount
{/* 定义存储**总账_数据结构**的变量及其类型

库存总账

物资	单位	在库数量	在库金额	在库单价
M1	张	80	74666.68	933.3333
M2	张	0	0.00	0.0000
M3	张	0	0.00	0.0000

*/
}
TransferFunc()
{ //出库记账处理：
//总账中的在库数量 = 总账中的在库数量 - 流水账中的出库数量
//总账中的在库金额 = 总账中的在库金额 - 流水账中的出库金额
//...
}
}

● “出库-过账”依存于“出库单-记账”，仅为其所用;
●有一个“出库单-记账”，就有一个“出库-过账”.

库存流水账(类1)
(出库单-记账)

Out-Bill { 出库单的数据结构 }
iAccount { 流水账的数据结构 }

Out-Func()
{ //将出库单转记流水账的程序 }

Tmp-iAccout { 临时流水账数据存储表 }

组合了

库存总账(类1)
(出库-过账)

iAccount { 流水账的数据结构 }
gAccount { 总账的数据结构 }

TransferFunc()
{ //出库进行过账的程序 }

iAccout { 流水账数据存储表 }

gAccout { 总账数据存储表 }

软件不同设计的艺术
(1)类的不同设计

入库单-记账-过账的紧耦合设计

- “入库-过账”依存于“入库单-记账”，仅为其所用;
- 有一个“入库单-记账”，就有一个“入库-过账”.

Class 入库单-记账

```
{
    Struct In-Bill
    { /* 定义存储入库单_数据结构的变量及其类型
```

入库单

入库单编号：_____ 仓库：_____

供应商：_____

物资名称	规格/型号	入库数量	计量单位	单价	总价	对应验收单号	说明
MAT1		200	包	1000.00	200,000.00		
MAT2		100	个	150.00	15,000.00		

采购员：_____ 质检员：_____ 保管员：_____ 日期：_____

```
    */
}
Struct iAccount
{ /* 定义存储流水账_数据结构的变量及其类型
```

库存流水账

日期	物资	入/出	数量	单价	金额
2012.06.01	M1	入	200	1,100.0000	220,000.00
2012.06.02	M1	出	80	1,100.0000	88,000.00
2012.06.03	M1	出	80	1,100.0000	88,000.00

```
    */
}
int In-Func( )
{
    //将 In-Bill 中的数据，赋值给 iAccount 中的变量;
    //将 iAccount 的数据写入临时流水账数据库中 Tmp-iAccount.
}
int InTransfer()
{
    //创建入库-过账对象。如 TransferObj = new 入库-过账;
    //将待过账的流水账数据传递给入库-过账对象 TransferObj.
    //调用 TransferObj.TranferFun()执行过账。
}
}
```

Class 入库-过账

```
{
    Struct iAccount
    { /* 定义存储流水账_数据结构的变量及其类型
```

库存流水账

日期	物资	入/出	数量	单价	金额
2012.06.01	M1	入	200	1,100.0000	220,000.00
2012.06.02	M1	出	80	1,100.0000	88,000.00
2012.06.03	M1	出	80	1,100.0000	88,000.00

```
    */
}
Struct gAccount
{ /* 定义存储总账_数据结构的变量及其类型
```

库存总账

物资	单位	在库数量	在库金额	在库单价
M1	张	80	74666.68	933.3333
M2	张	0	0.00	0.0000
M3	张	0	0.00	0.0000

```
    */
}
TransferFunc()
{  //入库记账处理:
   //总账中的在库数量 = 总账中的在库数量 + 流水账中的入库数量
   //总账中的在库金额 = 总账中的在库金额 + 流水账中的入库金额
   //...
}
}
```

库存流水账(类2)
(入库单-记账)

In-Bill { 入库单的数据结构 }
iAccount { 流水账的数据结构 }

In-Func()
{ //将入库单转记流水账的程序 }

Tmp-iAccout { 临时流水账数据存储表 }

组合了

库存总账(类2)
(入库-过账)

iAccount { 流水账的数据结构 }
gAccount { 总账的数据结构 }

TransferFunc()
{ //入库进行过账的程序 }

iAccout { 流水账数据存储表 }

gAccout { 总账数据存储表 }

软件不同设计的艺术
(1)类的不同设计

入/出库单-记账-过账的紧耦合设计

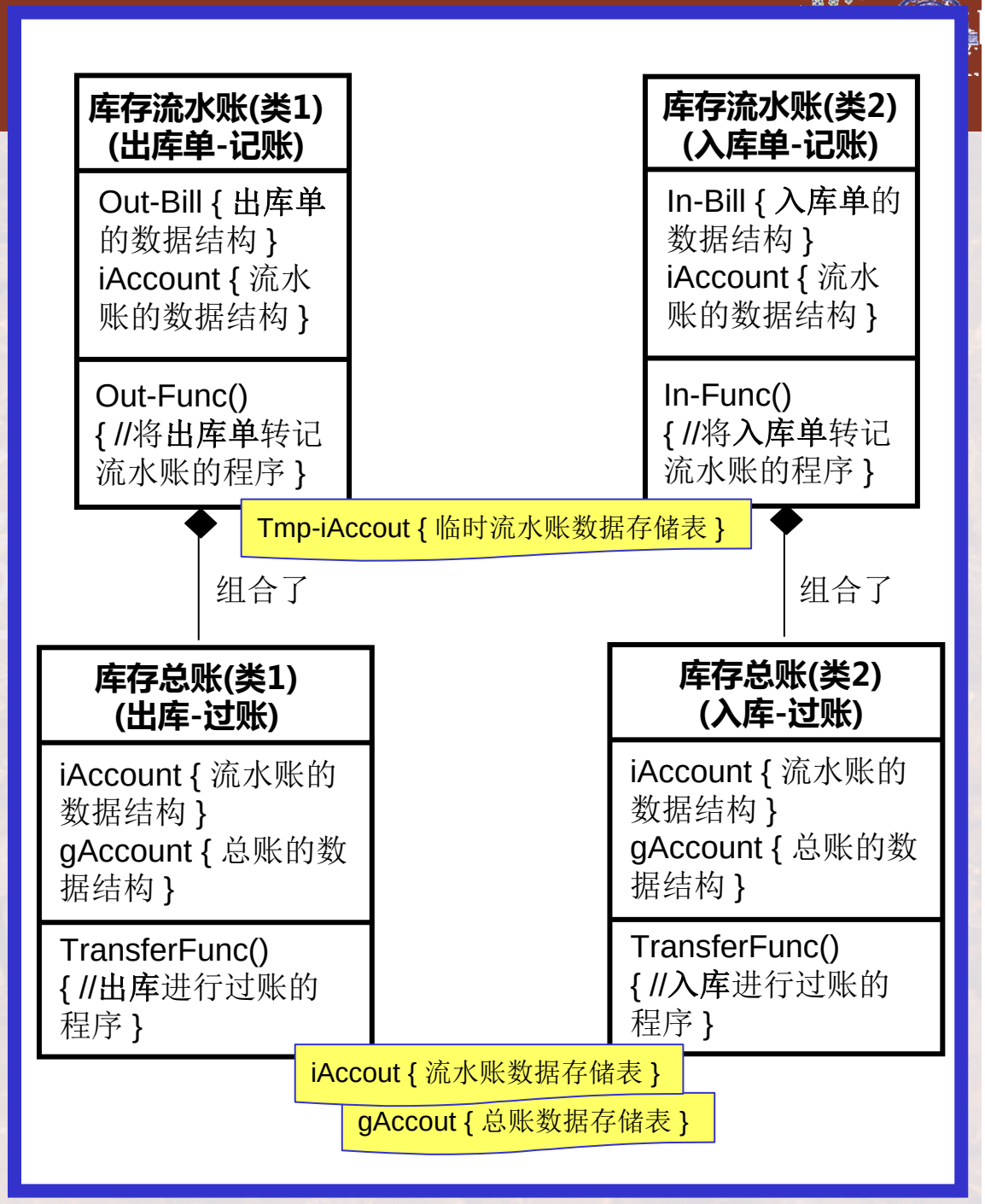
每种单据都对应一个 “过账” 程序

同一个库存总账，由多个程序修改，一致性 怎样呢?

出库是在库量减少

入库是在库量增加

有没有其他方案呢?



战德臣 教授

库存账(类)的三种不同设计方案的对比

战德臣 教授

入/出库单-记账-过账的松耦合设计

●多种单据的记账-过账共享同一个过账程序；
●单据类别增加，只需在一处改变程序，便可处理不同单据的过账

```
Class 过账
{   int BillType; //单据类型
    Struct iAccount
    { /* 定义存储流水账_数据结构的变量及其类型
```

库存流水账

日期	物资	入/出	数量	单价	金额
2012.06.01	M1	入	200	1,100.0000	220,000.00
2012.06.02	M1	出	80	1,100.0000	88,000.00
2012.06.03	M1	出	80	1,100.0000	88,000.00

```
    */
    }
    Struct gAccount
    { /* 定义存储总账_数据结构的变量及其类型
```

库存总账

物资	单位	在库数量	在库金额	在库单价
M1	张	80	74666.68	933.3333
M2	张	0	0.00	0.0000
M3	张	0	0.00	0.0000

```
    */
    }
    TransferFunc(BillType)
    { If BillType = "入库单"  then  { //入库记账处理：
        //总账中的在库数量 = 总账中的在库数量 + 流水账中的入库数量
        //总账中的在库金额 = 总账中的在库金额 + 流水账中的入库金额
        //…  }
      If BillType = "出库单"  then  { //出库记账处理：
        //总账中的在库数量 = 总账中的在库数量 – 流水账中的出库数量
        //总账中的在库金额 = 总账中的在库金额 – 流水账中的出库金额
        //…  }
      If BillType = "XXX"  then  { //XXX 记账处理：不同单据有不同的记账规则
        //总账中的在库数量 = 总账中的在库数量 ? 流水账中的 XXX 数量
        //总账中的在库金额 = 总账中的在库金额 ? 流水账中的 XXX 金额
        //…  }
    }
}
```

```
Class 入库单-记账
{
    Struct In-Bill
    { /* 定义存储入库单_数据结构的变量及其类
```

入库单

入库单编号：_____ 仓库：_____
供应商：

物资名称	规格/型号	入库数量	计量单位	单价
MAT1		200	包	1000.00
MAT2		100	个	150.00

采购员：_____ 质检员：_____ 保管

```
    */
    }
    Struct iAccount
    { /* 定义存储流水账_数据结构的变量及其类
```

库存流水账

日期	物资	入/出	数量	单价
2012.06.01	M1	入	200	1,100.0000
2012.06.02	M1	出	80	1,100.0000
2012.06.03	M1	出	80	1,100.0000

```
    */
    }
    int In-Func( )
    {
        //将 In-Bill 中的数据，赋值给 iAccount 中
        //将 iAccount 的数据写入临时流水账数据
    }
    int InTransfer()
    {
        //创建出库-过账对象。如 TransferObj = new 过账；
        //将待过账的流水账数据传递给过账对象 TransferObj.
        //设置单据类型为 BillType = "入库单"
        //调用 TransferObj. TranferFun(BillType)执行过账。
    }
}
```

```
Class 出库单-记账
{
    Struct Out-Bill
    { /* 定义存储出库单_数据结构的变量及其类型
```

出库单

出库单编号：_____ 仓库：_____
会计：_____ 用途：_____

物资名称	规格/型号	出库数量	计量单位	单价	总价	领料单号/提货单号	说明
MAT1		200	包	1000.00	200,000.00		
MAT2		100	个	150.00	15,000.00		

保管员：_____ 领料员：_____ 日期：_____

```
    */
    }
    Struct iAccount
    { /* 定义存储流水账_数据结构的变量及其类型
```

库存流水账

日期	物资	入/出	数量	单价	金额
2012.06.01	M1	入	200	1,100.0000	220,000.00
2012.06.02	M1	出	80	1,100.0000	88,000.00
2012.06.03	M1	出	80	1,100.0000	88,000.00

```
    */
    }
    int Out-Func( )
    {
        //将 Out-Bill 中的数据，赋值给 iAccount 中的变量；
        //将 iAccount 的数据写入临时流水账数据库中 Tmp-iAccount.
    }
    int OutTransfer()
    {
        //创建出库-过账对象。如 TransferObj = new 过账；
        //将待过账的流水账数据传递给过账对象 TransferObj.
        //设置单据类型为 BillType = "出库单"
        //调用 TransferObj. TranferFun(BillType)执行过账。
    }
```

软件不同设计的艺术
(1)类的不同设计

战德臣 教授

入/出库单-记账-过账的松耦合设计

两个“过账”程序变成一个“过账”程序，怎么处理不同的单据呢?

库存流水账(类1)(出库单-记账)

Out-Bill { 出库单的数据结构 }
iAccount { 流水账的数据结构 }

Out-Func()
{ //将出库单转记流水账的程序 }

BillType=‘出库单’

库存流水账(类2)(入库单-记账)

In-Bill { 入库单的数据结构 }
iAccount { 流水账的数据结构 }

In-Func()
{ //将入库单转记流水账的程序 }

BillType=‘入库单’

Tmp-iAccout { 临时流水账数据存储表 }

库存总账(类)(过账)

iAccount { 流水账的数据结构 }
BillType; //单据类型
gAccount { 总账的数据结构 }

TransferFunc()
{ //按单据类型调用不同的过账处理程序，处理数据后统一存储 }

BillType?

BillType=‘入库单’ → In-Bill-Transfer()
{ //入库单据的过账函数 }

BillType=‘出库单’ → Out-Bill-Transfer()
{ //出库单据的过账函数 }

BillType=‘XX单据’ → XX-Bill-Transfer()
{ //XX单据的过账函数 }

iAccout { 流水账数据存储表 }

gAccout { 总账数据存储表 }

单据类别增加，只需在一处改变程序

战德臣 教授

库存账(类)的三种不同设计方案的对比

战德臣 教授

将不同类别单据的过账规则存储于数据表中，依据规则过账

过账规则表

单据类别	在库数量	在库金额	说明
入库单	+	+	当单据为入库单时， 在库数量=在库数量 + 过账数量； 在库金额=在库金额 + 过账金额；
出库单	-	-	当单据为出库单时， 在库数量=在库数量 - 过账数量； 在库金额=在库金额 - 过账金额；
盘盈单	+		当单据为盘盈单时， 在库数量=在库数量 + 过账数量； 在库金额不变
盘亏单	-		当单据为盘亏单时， 在库数量=在库数量 - 过账数量； 在库金额不变
《新类别单据》			

```
Class 过账
{   int BillType; //单据类型
    Struct iAccount
    { /* 定义存储流水账_数据结构的变量及其类型
    */
    }
    Struct gAccount
    { /* 定义存储总账_数据结构的变量及其类型
    */
    }
    TransferFun(BillType)
    {
        // QtyRule = 获取在库数量的过账规则(BillType)
        //SumRule = 获取在库金额的过账规则(BillType)
      If QtyRule = "+" then {
        //总账中的在库数量 = 总账中的在库数量 + 过账数量
        //... }
      If QtyRule = "-" then {
        //总账中的在库数量 = 总账中的在库数量 - 过账数量
        //... }
      If SumRule = "+" then {
        //总账中的在库金额 = 总账中的在库金额 + 过账金额
        //... }
      If SumRule = "-" then {
        //总账中的在库金额 = 总账中的在库金额 - 过账金额
        //... }
    }
}
```

```
Class 入库单-记账
{
    Struct In-Bill
    { /* 定义存储入库单_数据结构的变量及其类
    */
    }
    Struct iAccount
    { /* 定义存储流水账_数据结构的变量及其类
    */
    }
    int In-Func( )
    {
        //将 In-Bill 中的数据，赋值给 iAccount 中
        //将 iAccount 的数据写入临时流水账数据库中 Tmp-iAccount.
    }
    int InTransfer()
    {
        //创建出库-过账对象。如 TransferObj = new 过账；
        //将待过账的流水账数据传递给过账对象 TransferObj.
        //设置单据类型为 BillType = "入库单"
        //调用 TransferObj. TranferFun(BillType)执行过账。
    }
}
```

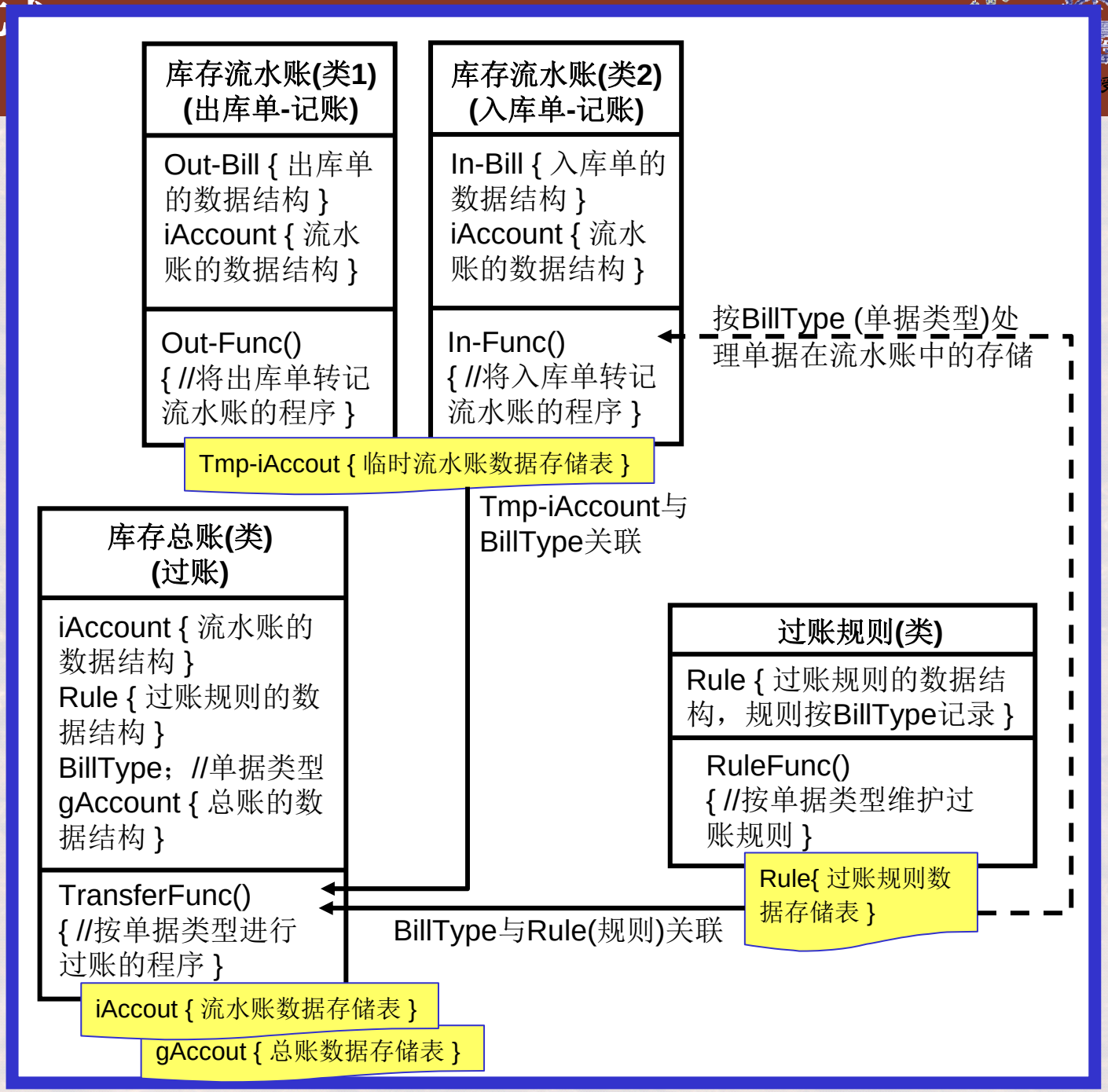
```
Class 出库单-记账
{
    Struct Out-Bill
    { /* 定义存储出库单_数据结构的变量及其类型
    */
    }
    Struct iAccount
    { /* 定义存储流水账_数据结构的变量及其类型
    */
    }
    int Out-Func( )
    {
        //将 Out-Bill 中的数据，赋值给 iAccount 中的变量；
        //将 iAccount 的数据写入临时流水账数据库中 Tmp-iAccount.
    }
    int OutTransfer()
    {
        //创建出库-过账对象。如 TransferObj = new 过账；
        //将待过账的流水账数据传递给过账对象 TransferObj.
        //设置单据类型为 BillType = "出库单"
        //调用 TransferObj. TranferFun(BillType)执行过账。
    }
}
```

软件不同设计的艺术
(1)类的不同设计

将不同类别单据的过账规则存储于数据表中，依据规则过账

有一个规则维护程序，维护一个规则表，记录不同类别单据的过账规则。过账程序依据规则进行过账。

单据类别增加，不需改变程序，便可处理不同单据过账



战德臣 教授

库存账(类)的三种不同设计方案的对比

软件不同设计的艺术
示例2：不同软件间连接的不同设计

战德臣

哈尔滨工业大学　教授.博士生导师

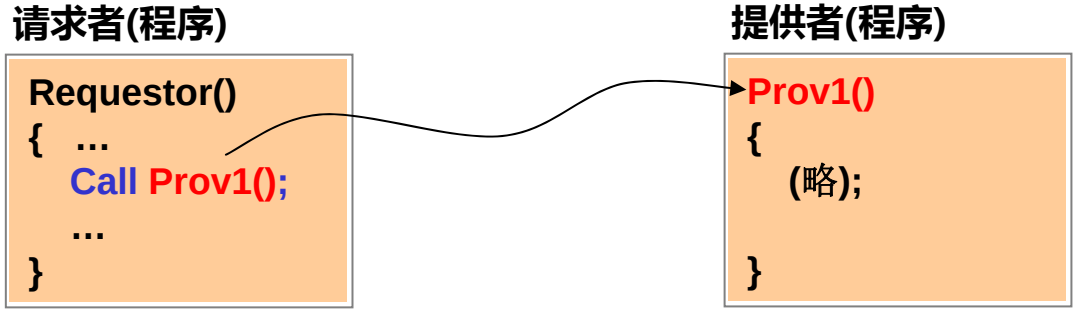
教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology

战德臣 教授

直接连接式集成的软件体系结构

一个服务中心系统和一个车队系统的自动连接需求示意



请求者和提供者实现紧耦合的连接(程序调用示意)

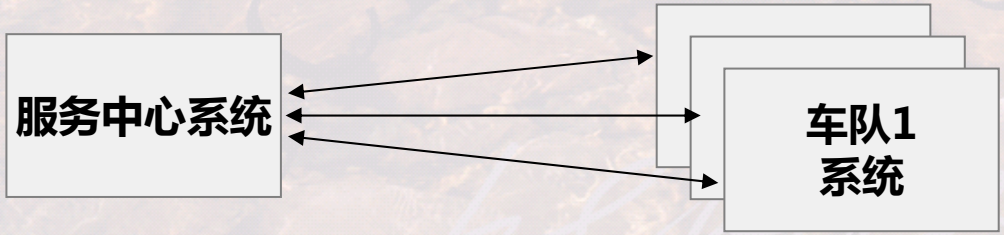
- 请求者需要知道提供者相应的函数名；
- 提供者函数名发生变化，则请求者的程序也需要随之改变，否则不能正确集成

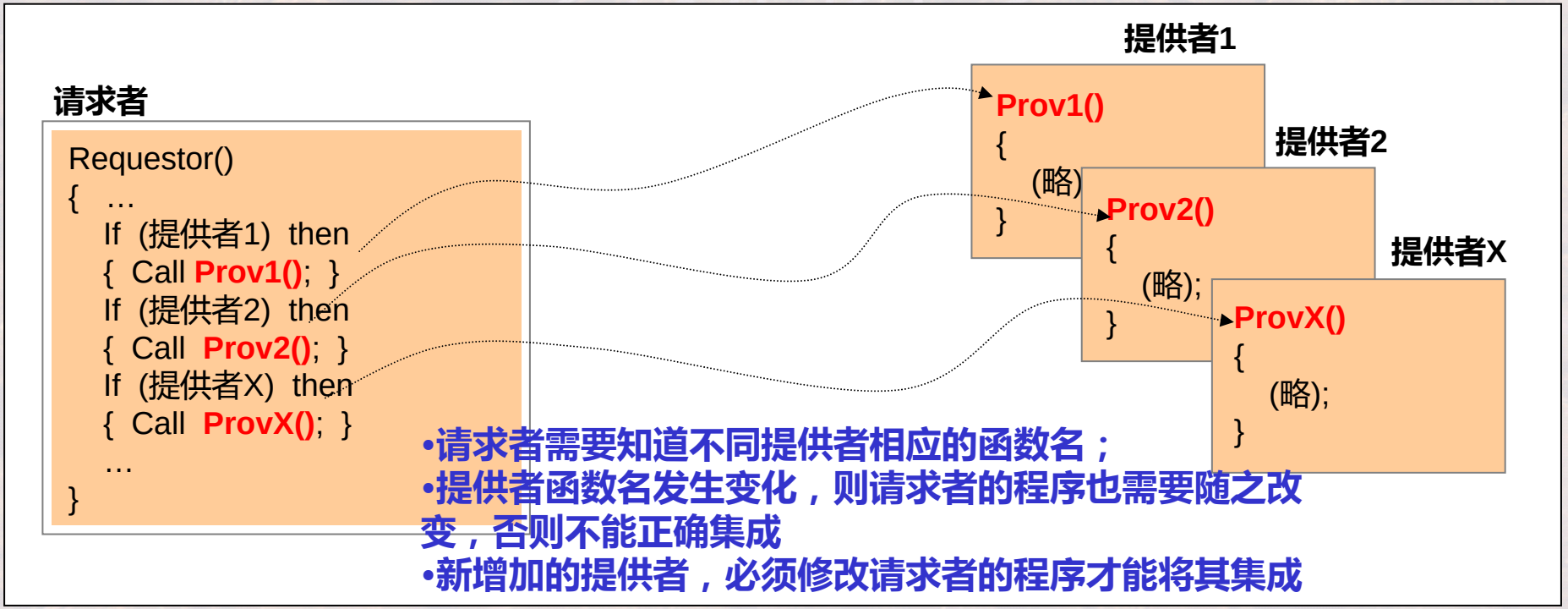
请求者和提供者实现紧耦合的连接(简化示意)

战德臣 教授

直接连接式集成的软件体系结构

服务中心系统希望实现和多个车队系统的任意自动连接需求示意

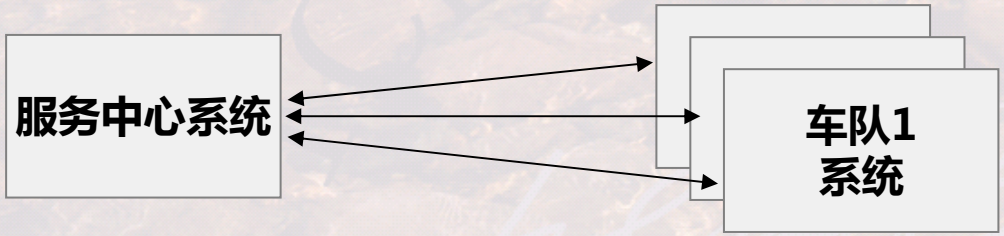


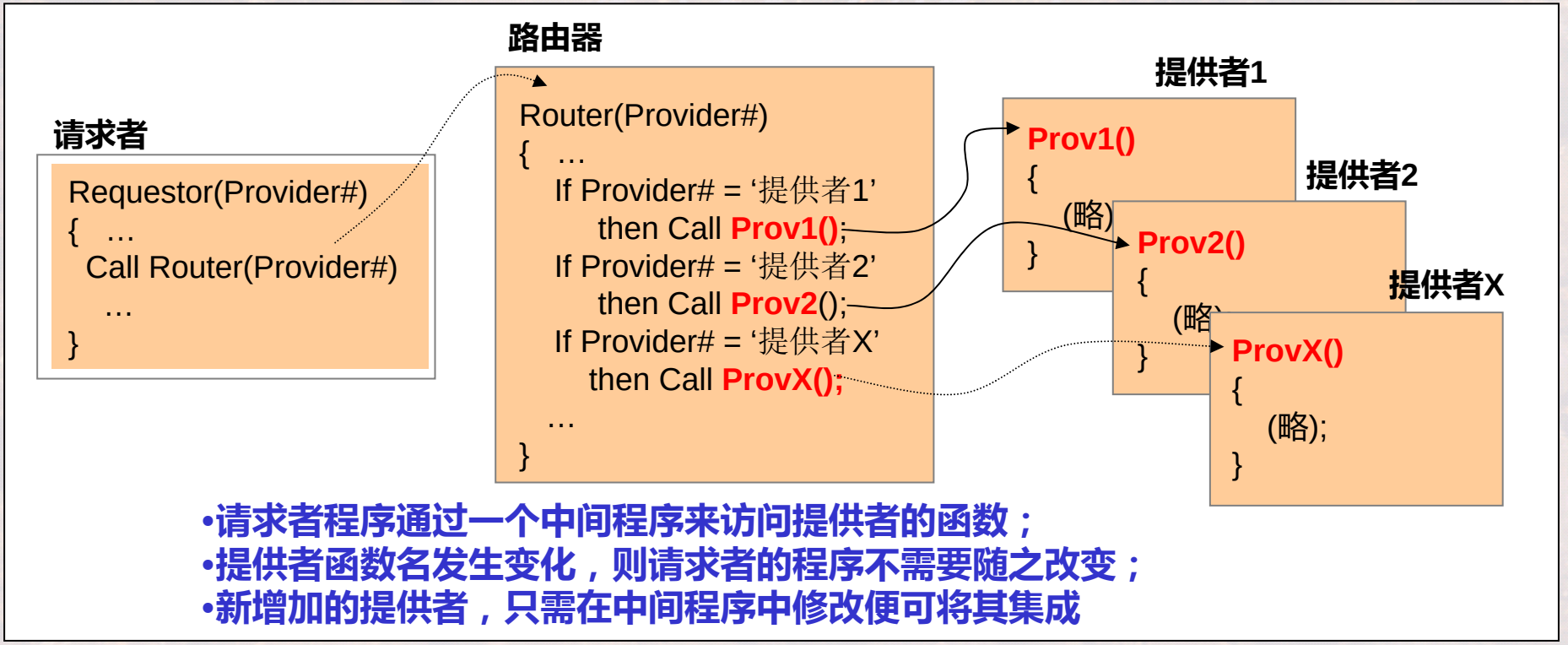


战德臣 教授

间接连接式集成的软件体系结构

服务中心系统希望实现和多个车队系统的任意自动连接需求示意





- 请求者程序通过一个中间程序来访问提供者的函数；
- 提供者函数名发生变化，则请求者的程序不需要随之改变；
- 新增加的提供者，只需在中间程序中修改便可将其集成

战德臣 教授

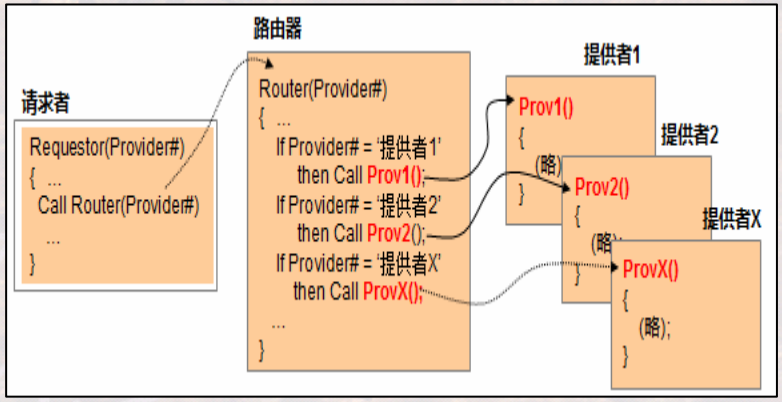
基于中间件集成的软件体系结构

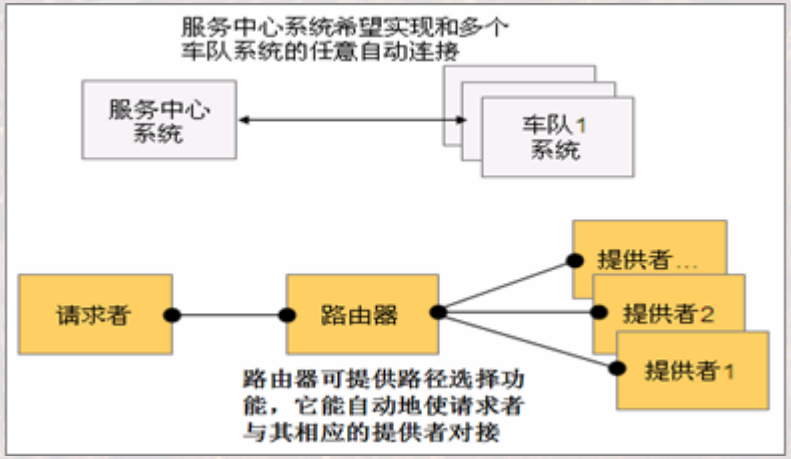
所谓的路由器是什么？其功能是什么？

◆**表示与存储**：存储了一系列的路径选择规则(即什么条件，选择什么路径)

◆**执行**：按请求者给出的条件，查找规则，根据找到的规则，确定相应的路径即提供者，实现二者的连接。

◆**转换**：必要的话要进行格式的转换。不同提供者有不同的信息格式，请求者有统一的格式，与不同提供者连接时，由路由器进行相应格式的转换。

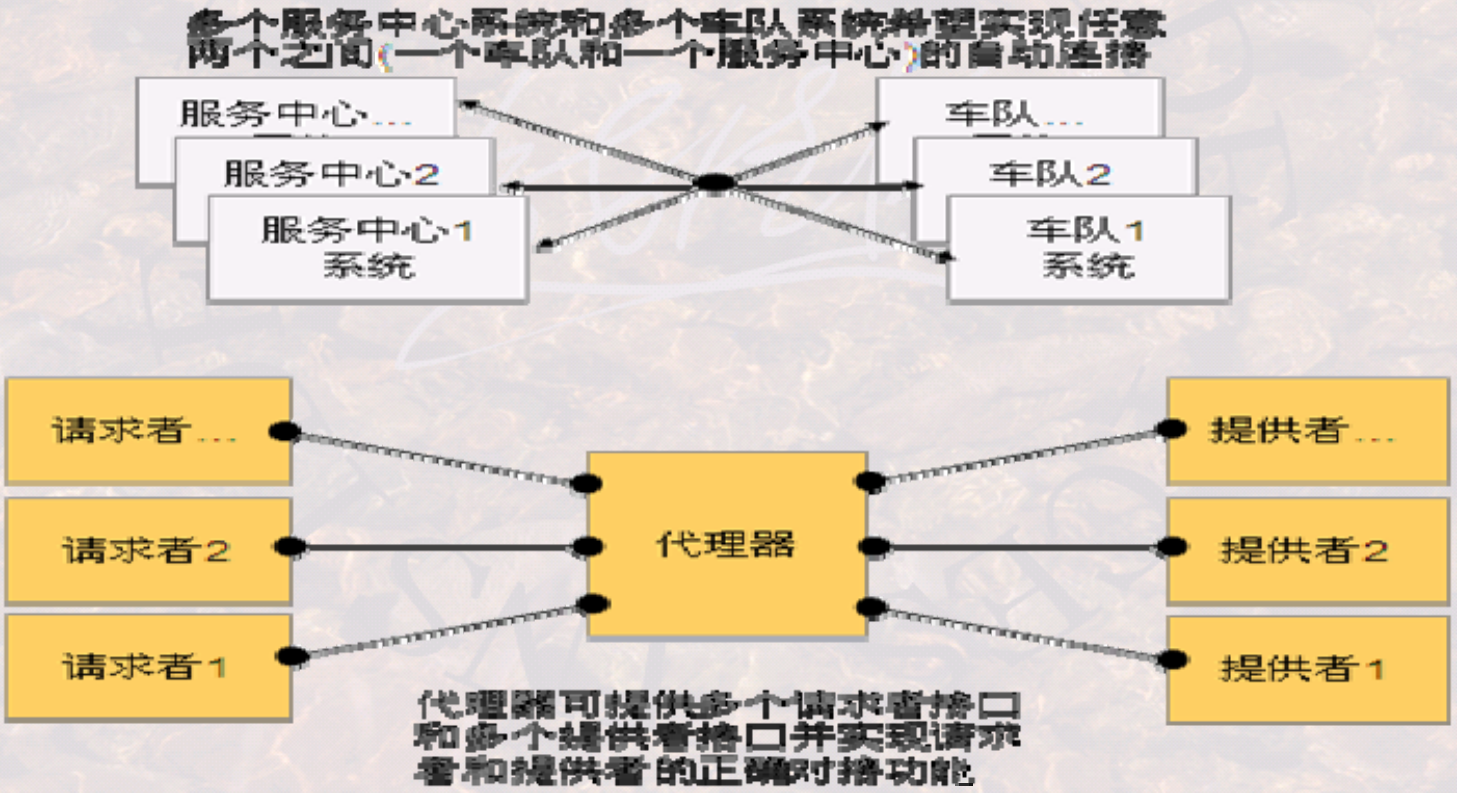




战德臣 教授

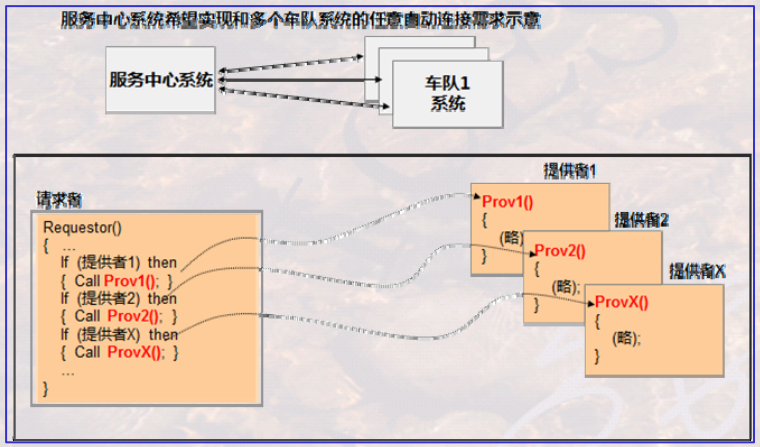
基于中间件集成的软件体系结构

◆想象一下，代理器应具有什么样的功能呢？

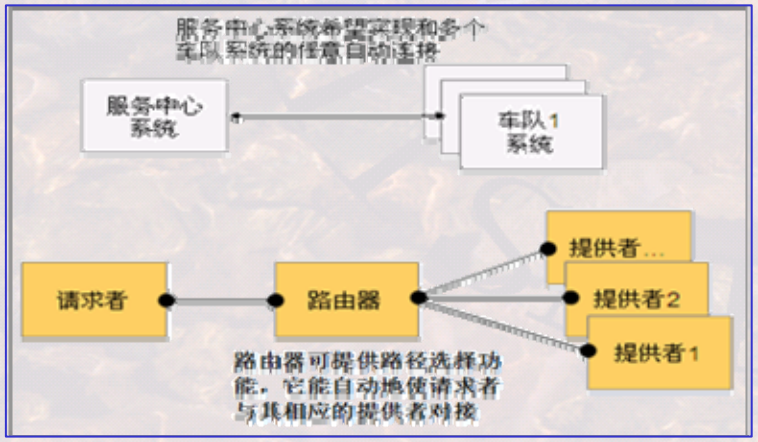


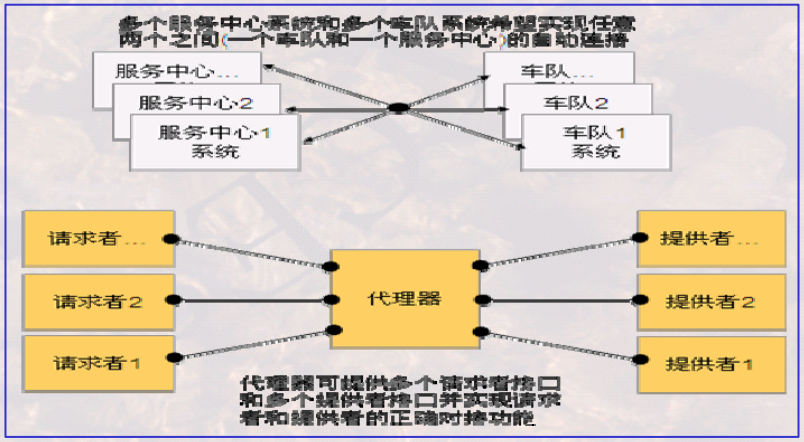
软件不同设计的艺术
(2)“集成”的不同设计

战德臣 教授



软件设计是否是艺术呢?
技术+艺术?





软件不同设计的艺术
软件体系结构与软件设计模式

战德臣

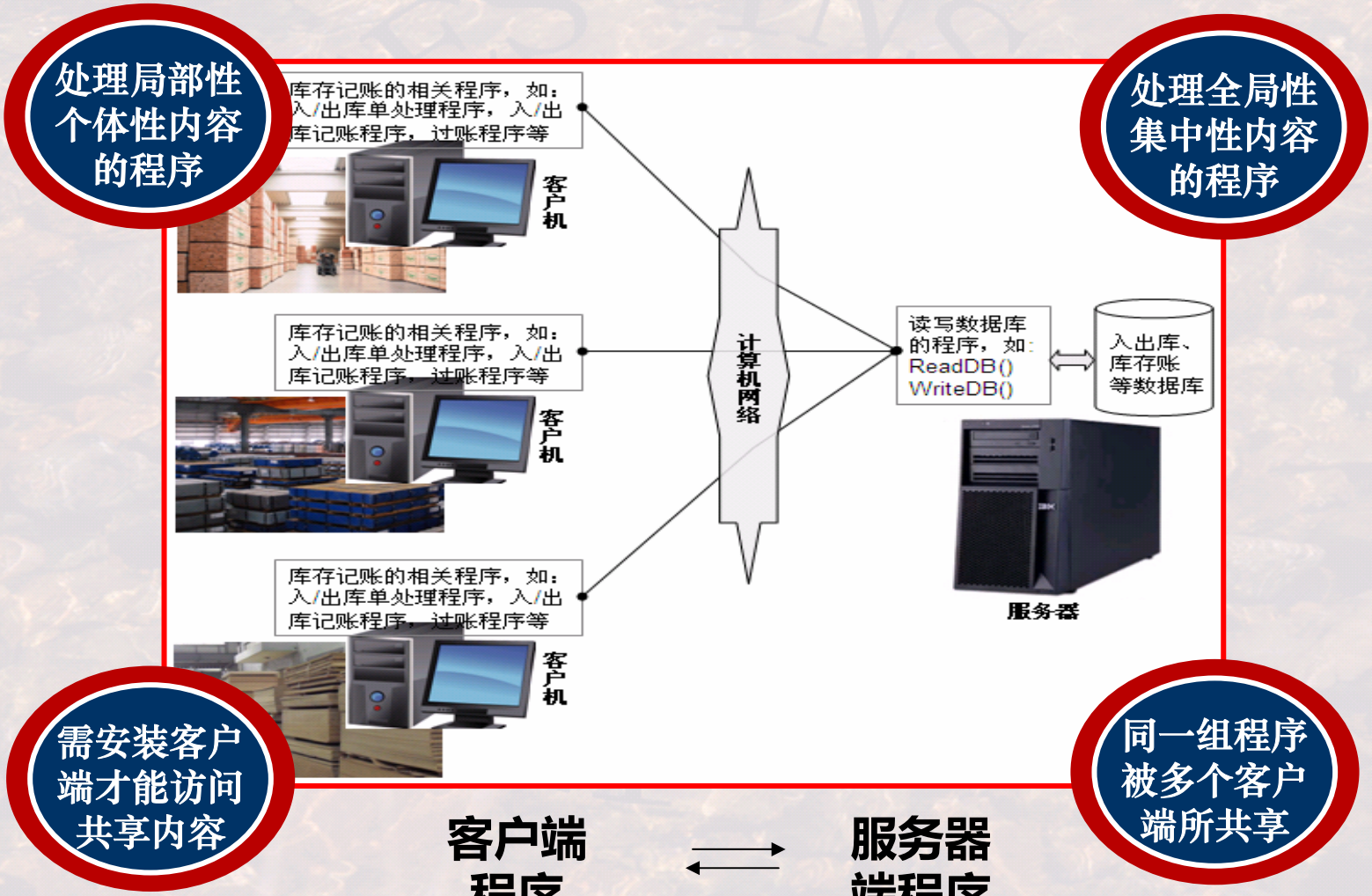
哈尔滨工业大学　教授.博士生导师

教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology

战德臣 教授

Client/Server结构—C/S结构



战德臣 教授

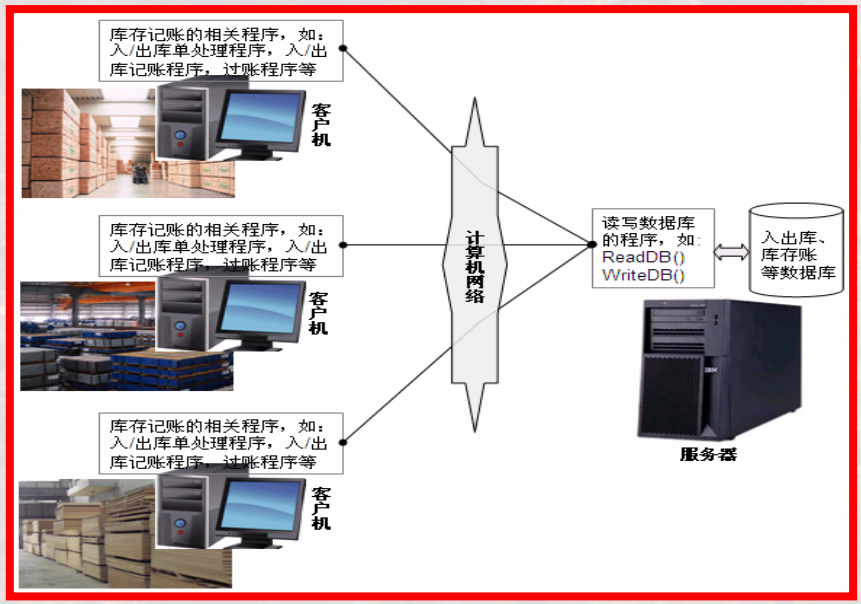
Browser/Server结构—B/S结构

通用浏览器 ⟷ Web程序 ⟷ 服务器端程序

软件不同设计的艺术
(3)不同的软件体系结构设计

战德臣 教授

Client/Server结构—C/S结构



客户端程序 ⟷ 服务器端程序

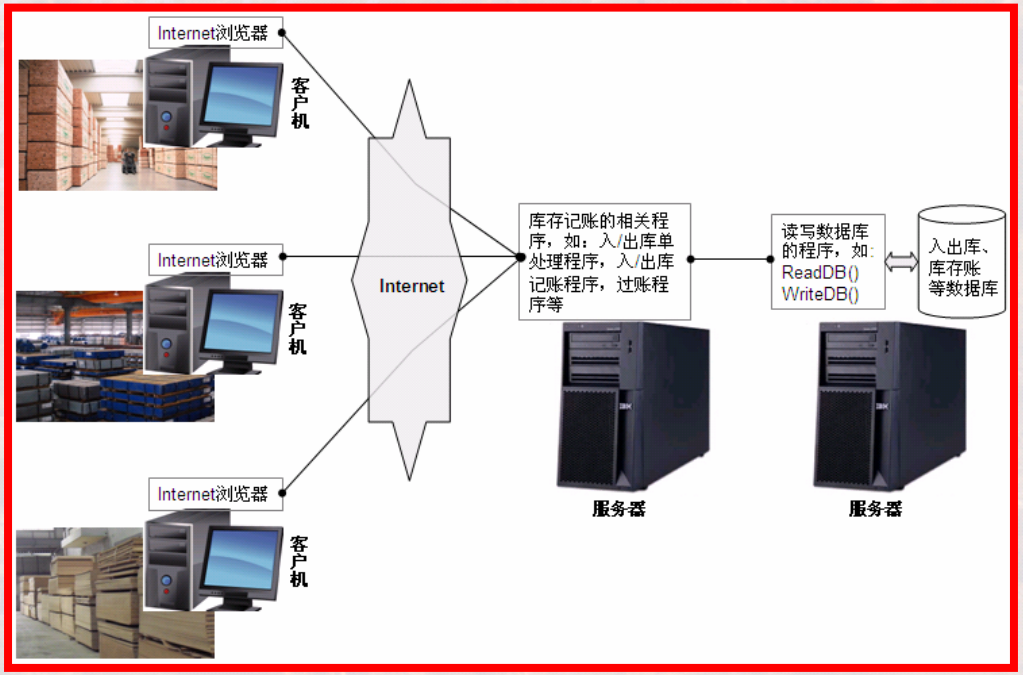
有限的客户机（装载客户端程序）

更新时要到每一处客户机进行更新

无限的客户机（只要有通用浏览器）

更新时只需在服务器上进行更新

Browser/Server结构—B/S结构



通用浏览器 ⟷ Web程序 ⟷ 服务器端程序

战德臣 教授

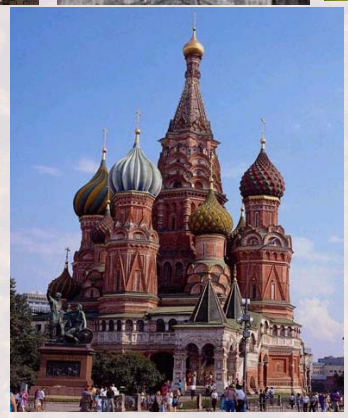
软件体系结构 vs. 建筑体系结构

◆构建软件也和建造不同特色建筑一样，虽然看不见，但也有不同的风格。建筑的不同风格，不仅体现在建筑的外观上，还体现在其基本的构件、构件间的连接方式、建造比例等，即体现的是一种体系结构。

◆同样，软件也有不同的体系结构，不同体系结构能够解决不同类别的问题。








战德臣 教授

软件体系结构

●**构件**：一组基本的构成要素

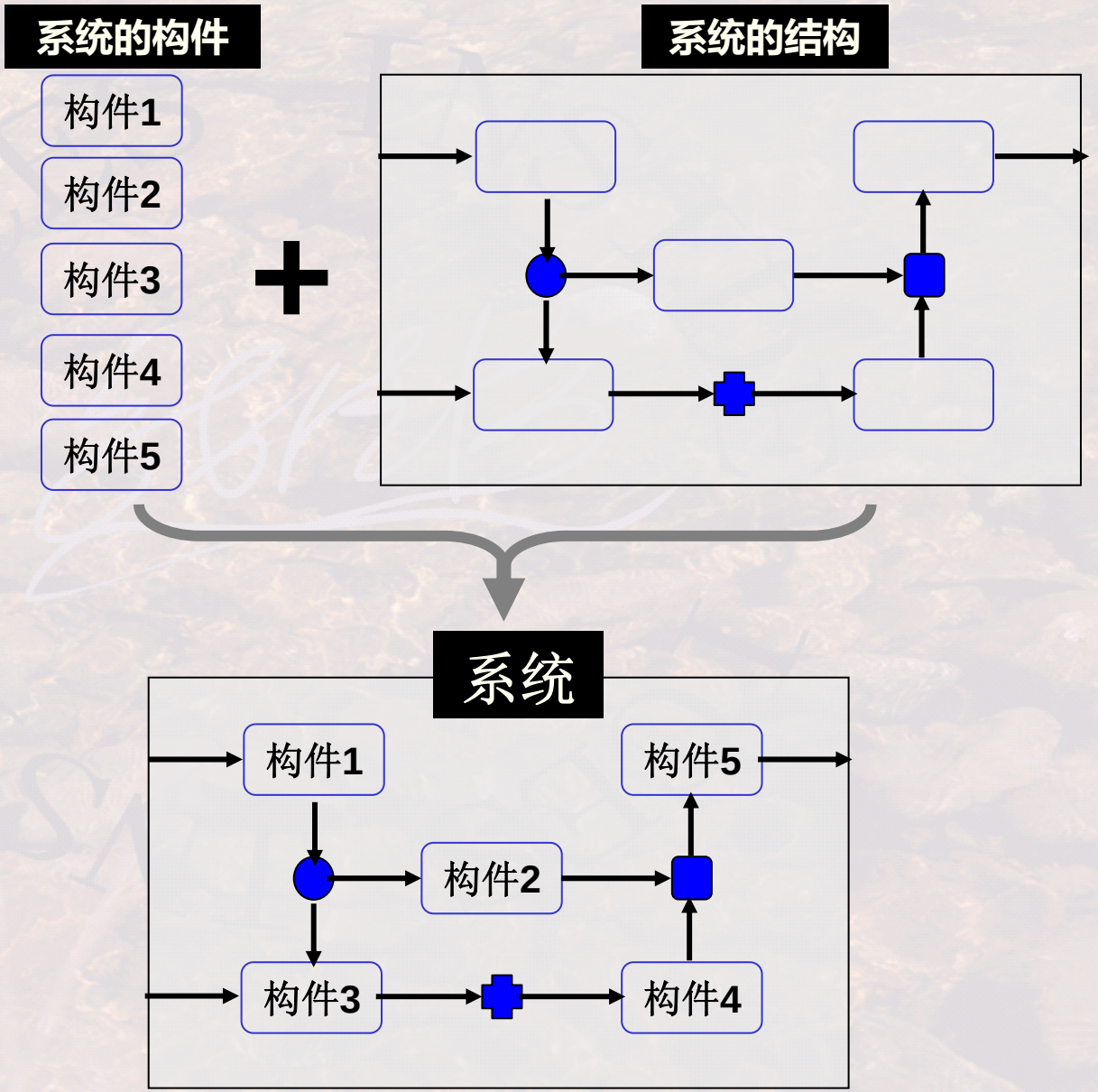
●**连接件**：这些要素之间的连接关系

●**物理分布**：这些要素连接之后形成的拓扑结构

●**约束**：作用于这些要素或连接关系上的限制条件

●**质量**：(运行时的)性能。

◆基于软件体系结构的开发



战德臣 教授

常见的软件体系结构和软件模式

◆常见的软件体系结构风格

✓ 数据流风格(如管道/过滤器、批处理等)、调用-返回风格(如主程序-子程序结构、面向对象结构、层次结构等)、仓库风格(如数据中心、黑板系统等)。

✓多处理器结构、客户机-服务器结构(C/S)、浏览器-服务器结构(B/S)、分布式对象结构(如总线结构)、面向服务的体系结构SOA

◆常见的软件设计模式(微观的软件体系结构)

✓ Factory Pattern(抽象工厂模式)、Façade Pattern(外观模式)、Command Pattern(命令模式)、Strategy Pattern(策略模式)、Iterator Pattern(迭代器模式)、Adaptor Pattern(适配器模式)、Observer Pattern(观察者模式)、Bridge Pattern(桥接模式)、Singleton Pattern(单件模式)等等

需要在后续课程中深入学习和掌握这些模式和体系结构

软件体系结构
软件设计模式

软件设计的本质是抽象与构造

战德臣

哈尔滨工业大学　教授.博士生导师

教育部大学计算机课程教学指导委员会委员

Research Center on Intelligent Computing for Enterprises & Services, Harbin Institute of Technology

战德臣 教授

抽象-理论-设计

◆抽象--是感性认识世界的手段。理论和设计的前提都需要抽象，没有抽象二者都是没有办法达成目标的(抽象的价值)

◆设计--是指构造软件系统来改造世界的手段，是工程的主要内容，只有设计才能造福于人类(设计的价值)

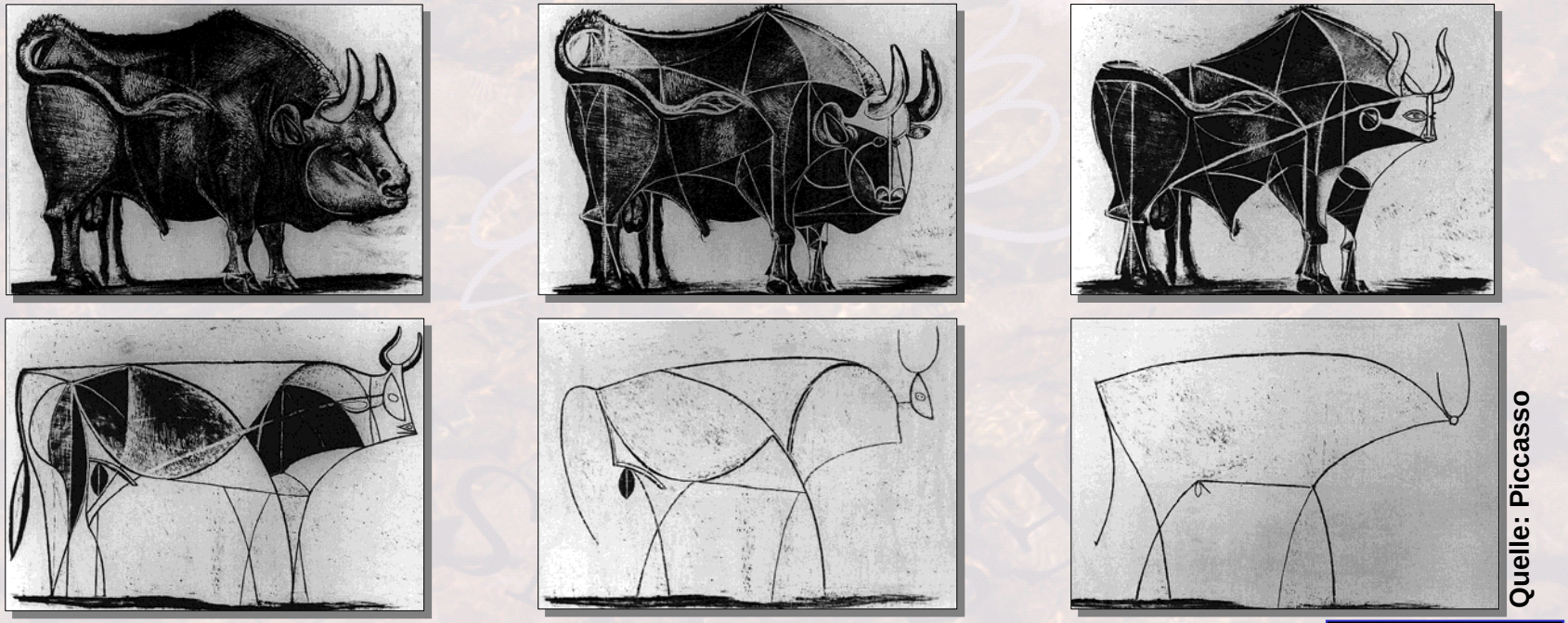
◆理论--是发现世界规律的手段，理论，如果不能指导设计，则是反映不出其价值的；设计，如果没有理论指导，则设计的严密性、可靠性、正确性是没有保证的(理论的价值)

软件设计的本质是抽象与构造
(2)什么是抽象？

战德臣 教授

抽象：是指由具体事物中发现其本质性特征和方法的过程。

具体化



Quelle: Piccasso

抽象化

抽象就是“逐渐剥离语义、逐渐去掉细节”

软件设计的本质是抽象与构造
(2)什么是抽象？

战德臣 教授

抽象：就是寻找相同的形式，处理可变的内容

“理解 → 区分 → 命名 → 表达”

学生登记表

学号	姓名	性别	出生年月	入学日期	家庭住址
98110101	张三	男	1980.10	1998.09	黑龙江省哈尔滨市
98110102	张四	女	1980.04	1998.09	吉林省长春市
98110103	张五	男	1981.02		哈尔市
98110201	王三	男	1980		

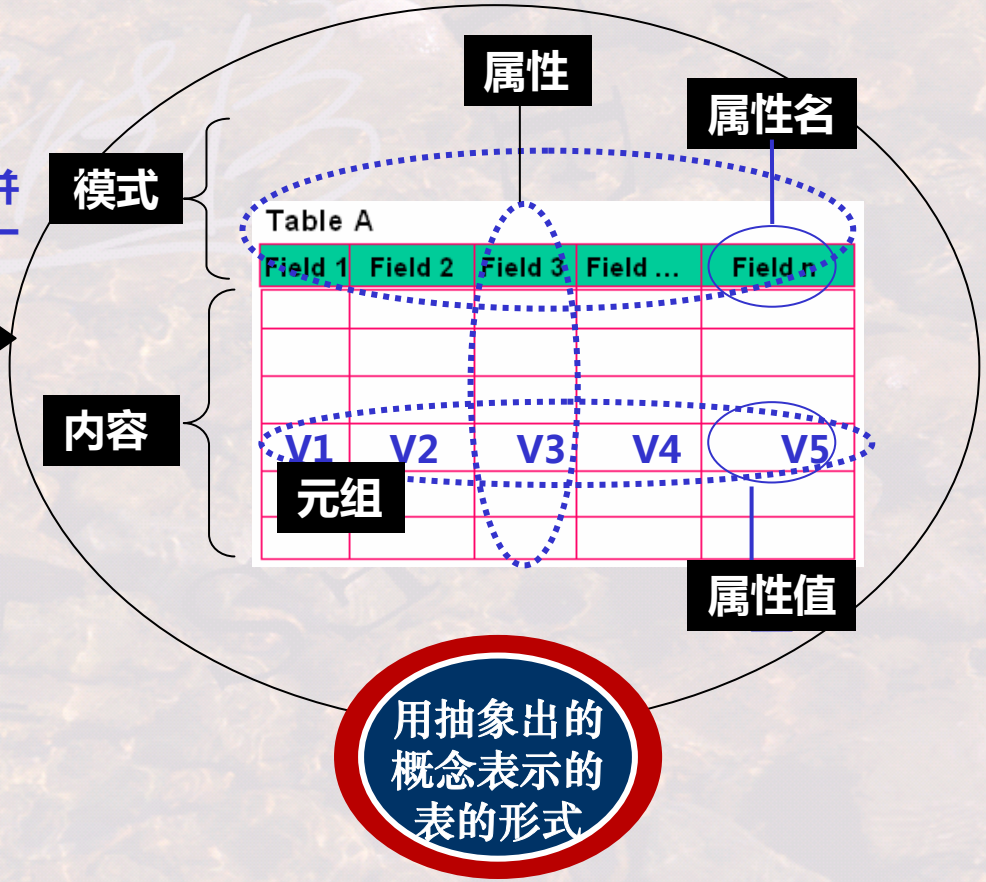
学生成绩单

班级	课程					
981101	数据库	李				00
981101	数据库	李四				90
981101	数据库	李四	98秋	98110101	张三	80
981101	计算机	李五	98秋	98110101	张三	89
981101	计算机	李五	98秋	98110102	张四	98
981101	计算机	李五	98秋	98110103	张五	72
981102	数据库	李四	99秋	98110201	王三	30
981102	数据库	李四	99秋	98110202	王四	90
981102	数据库	李四	99秋	98110203	王武	78

.....

若干具有相似性的具体的表

抽象：区分并命名表的每一个形式要素

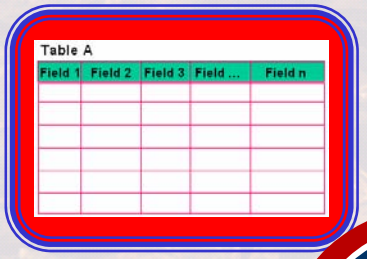


软件设计的本质是抽象与构造
(3)什么是设计?

战德臣 教授

设计：是构建软件系统的过程，是技术、原理在软件系统中实现的过程

“形式→构造→自动化”



可创建和操作任一表的语言

```
CREATE  TABLE 表名(列名1  类型  [NOT NULL]
        [, 列名2  类型  [NOT NULL]].....);
-------------------------------------------
SELECT  [DISTINCT]  列名1[, 列名2…]
FROM  表名1[, 表名2…]
[WHERE  条件1]
[GROUP BY  列名i1  [, 列名i2 …] [HAVING  条件2]]
[ORDER BY  表达式1  [ASC / DESC]…]
```

呵! 我似乎明白了。我依据形式设计一种语言。用户使用这种语言表达需求。我设计系统按用户表达的予以执行

设计一种“语言”—广义的语言，让用户表达其对具体表的具体需求 ---形式 与 构造

用户使用语言可表达创建和操作具体的表

```
CREATE TABLE 学生(学号 char(8), 姓名 char(40), 年龄 integer )

SELECT 学号 FROM 学生;
SELECT 学号, 姓名  FROM 学生
WHERE  年龄>20;
```

软件设计的本质是抽象与构造
(3)什么是设计?

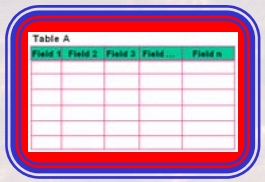
战德臣 教授

设计：是构建软件系统的过程，是技术、原理在软件系统中实现的过程

“形式→构造→自动化”

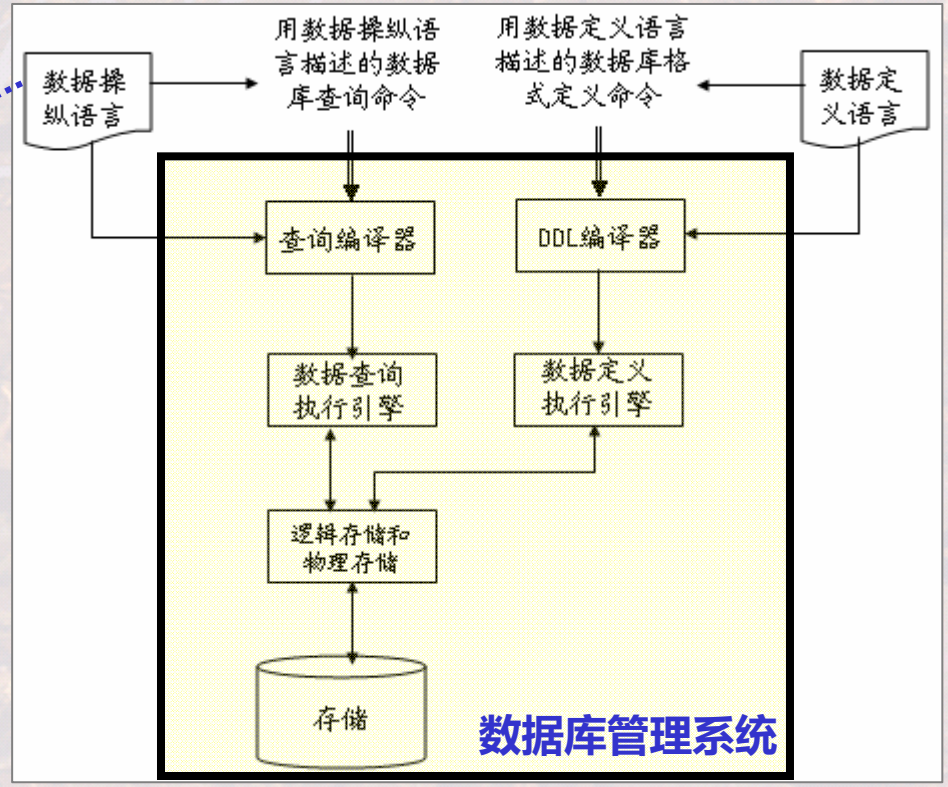
```
CREATE TABLE 学生(学号 char(8), 姓名 char(40), 年龄 integer )

SELECT 学号 FROM 学生;
SELECT 学号，姓名 FROM 学生
WHERE 年龄>20;
```



```
CREATE TABLE 表名(列名1 类型 [NOT NULL]
    [, 列名2 类型 [NOT NULL]].....);

SELECT [DISTINCT] 列名1[, 列名2…]
FROM 表名1[, 表名2…]
[WHERE 条件1]
[GROUP BY 列名i1 [, 列名i2 …] [HAVING 条件2]]
[ORDER BY 表达式1 [ASC / DESC]…]
```



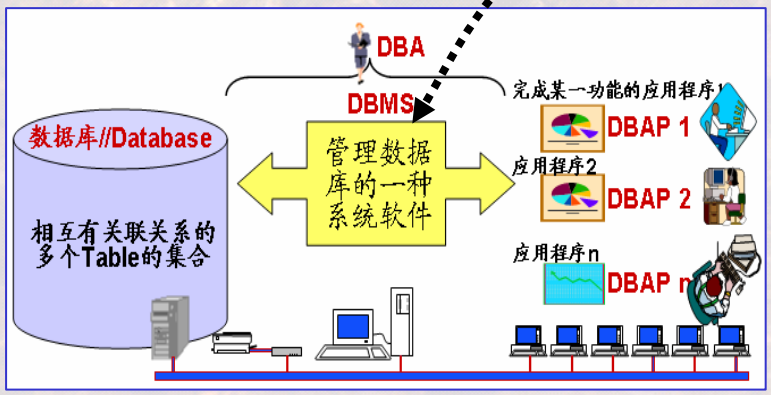
基于“语言”设计软件系统，自动执行用户用语言表达的需求

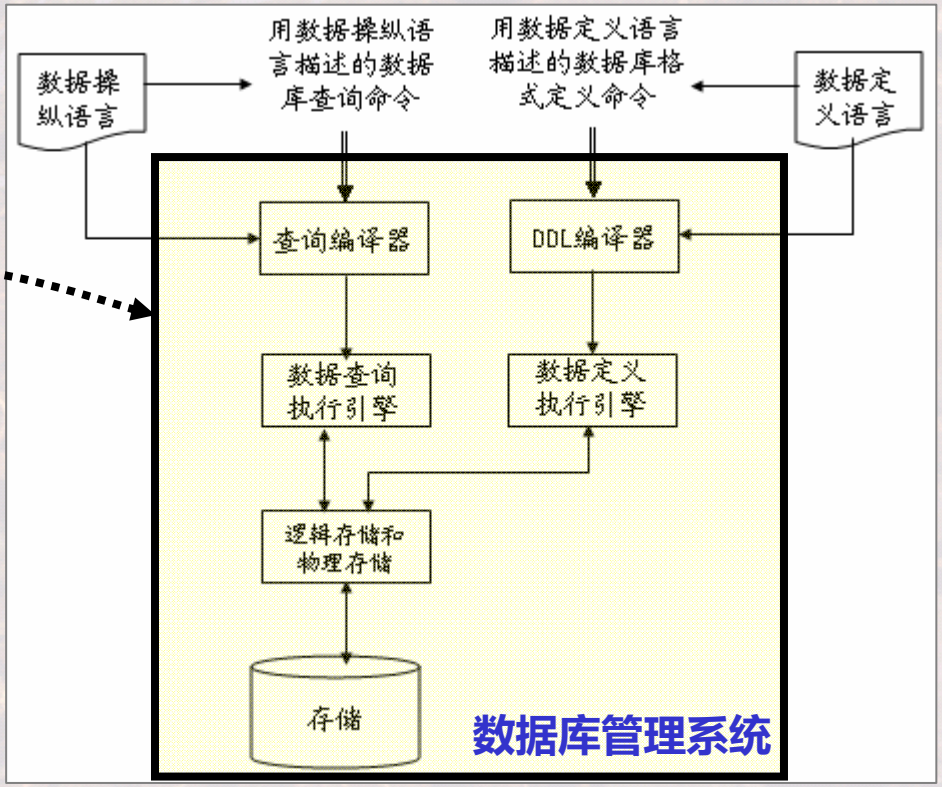
战德臣 教授

设计：是构建软件系统的过程，是技术、原理在软件系统中实现的过程

“形式→构造→自动化”

基于已实现的“系统”，再设计应用程序，--不断地自动化



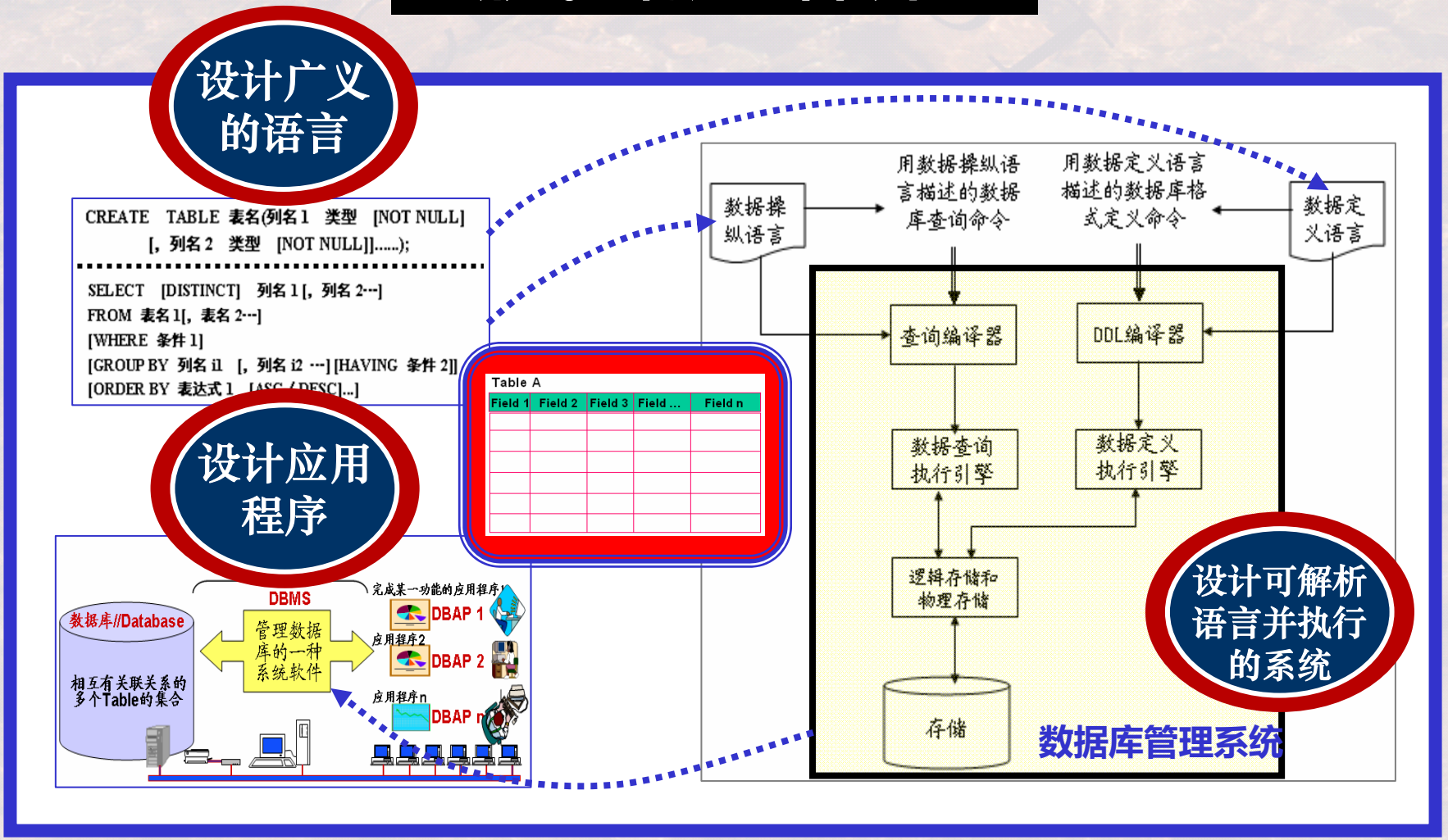


软件设计的本质是抽象与构造
(3)什么是设计?

战德臣 教授

设计：是构建软件系统的过程，是技术、原理在软件系统中实现的过程

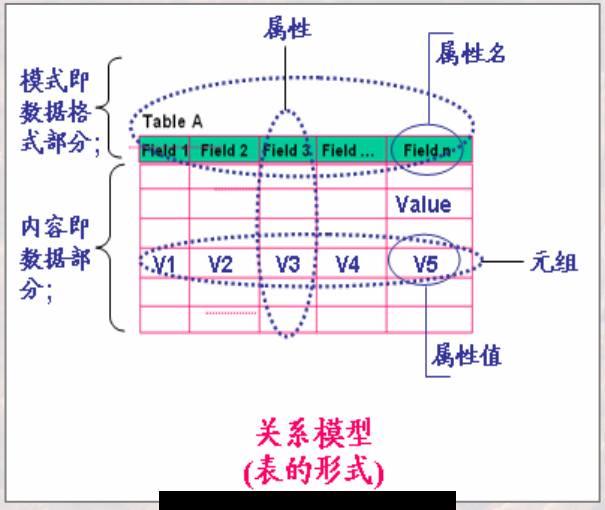
“形式→构造→自动化”



战德臣 教授

理论：对抽象和设计内容进行严格定义及严密化论证的过程。

“定义→性质(公理、定理)→证明”



抽象形态

理论：数学化逻辑严密化各种概念；

域(Domain)：一组值的集合

笛卡尔积(Cartesian Product)

$D_1 \times D_2 \times \ldots \times D_n = \{ (d_1, d_2, \ldots, d_n) \mid d_i \in D_i, i=1,\ldots,n \}$

关系(Relation)

一组域D1，D2，…，Dn的笛卡尔积的子集

$\cup$、$-$、$\times$、σ、π

$\pi_{S\#,Sname}(\sigma_{C\#="001"}(Student \bowtie SC))$

理论形态

- 将抽象形态区分出的概念，用数学语言进行严格定义
- 分析所定义概念的性质，形成公理
- 进一步分析相关的衍生的但重要的性质，形成定理
- 论证相关定理的正确性

战德臣 教授

理论：对抽象和设计内容进行严格定义及严密化论证的过程。

“定义→性质(公理、定理)→证明”

战德臣 教授

抽象结果的表达---用设计手段表达?用数学手段表达?

SC

S#	C#	Score
98030101	001	92.0
98030101	002	85.0

Student

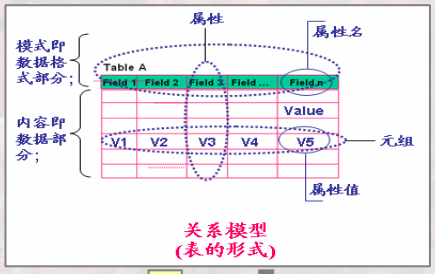
S#	Sname	Ssex	Sage	D#	Sclass
98030101	张三	男	20	03	980301

Course

C#	Cname	Chours	Credit	T#
001	数据库	40	6	001
003	数据结构	40	6	003
004	编译原理	40	6	001
005	C 语言	30	4.5	003
002	高等数学	80	12	004

抽象：区分并命名表的每一个形式要素

抽象



设计

先抽象再设计：从管理一个具体的表，到可管理所有的表

理论指导下的抽象：抽象更为严密

理论

域(Domain)：一组值的集合

笛卡尔积(Cartesian Product)

$D_1 \times D_2 \times \ldots \times D_n = \{ (d_1, d_2, \ldots, d_n) \mid d_i \in D_i, i=1,\ldots,n \}$

关系(Relation)

一组域D1，D2，…，Dn的笛卡尔积的子集

∪、－、×、σ、π

$\pi_{S\#,Sname}(\sigma_{C\#="001"}(Student \bowtie SC))$

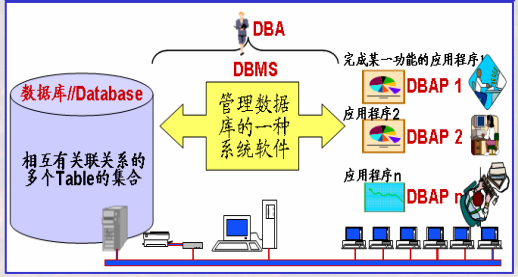
理论：数学化逻辑严密化各种概念；

理论支持设计：设计正确性、完备性判定方法

设计：语言/实现/系统

```
CREATE  TABLE 表名(列名1  类型  [NOT NULL]
        [, 列名2  类型  [NOT NULL]].....);

SELECT  [DISTINCT]  列名1[, 列名2…]
FROM  表名1[, 表名2…]
[WHERE  条件1]
[GROUP BY  列名i1  [, 列名i2 …] [HAVING  条件2]]
[ORDER BY  表达式1  [ASC / DESC]…]
```



第8讲 软件的艺术-软件设计