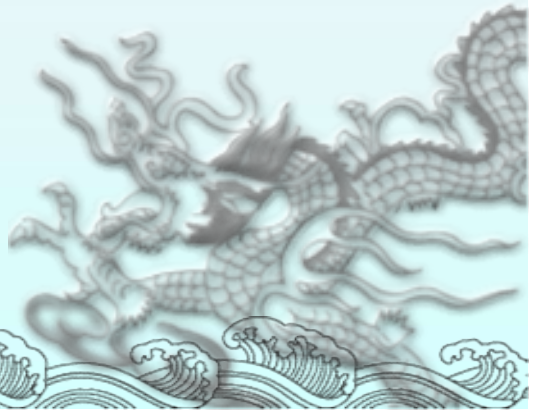
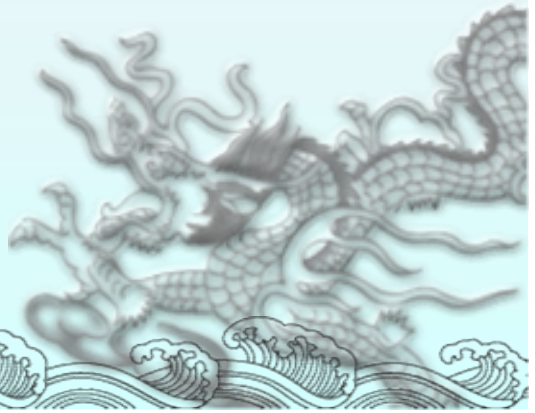
第5章 结构化软件设计



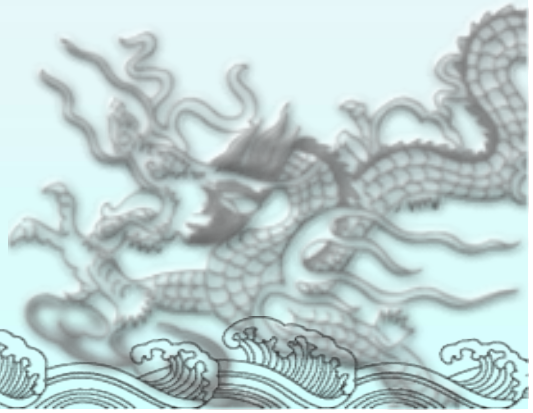
第5章 结构化软件设计

1）结构化设计的基本概念	掌握
2）方法和步骤	掌握
2）详细设计的方法。	掌握
3）软件设计的原则。	理解
4）影响软件设计的主要因素。	了解



内容提纲

- 结构化软件设计
  - 软件设计的基本概念
  - 软件设计原则和影响设计的因素
  - 结构化设计方法
  - 优化软件结构设计



5.1 软件设计的基本概念

5.1.1模块和模块化

- 一般把用一个名字就可调用的一段程序称为“模块”。模块具有如下三个基本属性。

功能：模块要完成的任务

↓

逻辑：模块内部执行过程

↓

状态：使用模块时的环境和条件

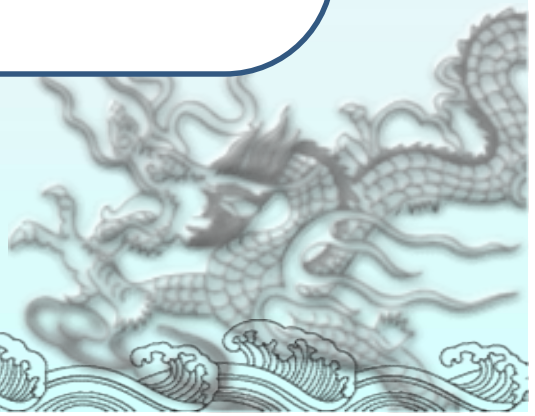
5.1.1模块和模块化（续）

模块化的优点：

1. 强调清楚地定义每个模块的功能和它的输入/输出参数；
2. 模块的实现细节隐藏在各自的模块之中，与其它模块之间的关系可以是调用关系；
3. 模块化程序易于调试和修改。

模块化的缺点：

1. 模块之间需要制定接口，模块越多，接口越复杂；
2. 模块越多，模块的集成成本越高。



5.1.1模块和模块化（续）

问题：怎样用较少的模块，较简单的接口，较简单的集成?

问题1：需要怎样划分模块?
问题2：怎样度量模块之间的关联?

5.1.2内聚和耦合

- 在软件设计中应该保持模块的独立性原则。反映模块独立性的有两个标准：内聚和耦合。

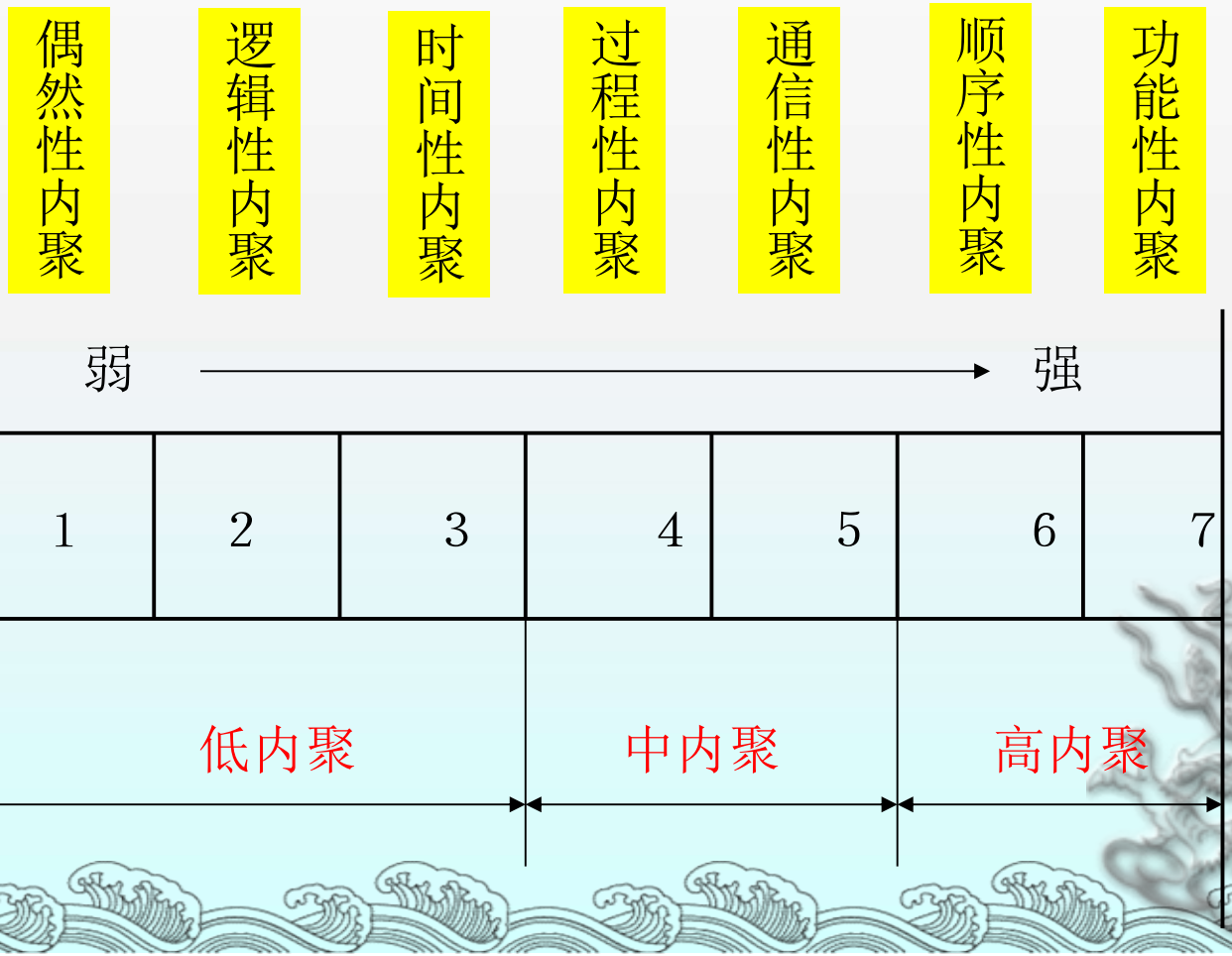
内聚：模块内部各个元素彼此结合的紧密程度

耦合：衡量模块之间彼此依赖的程度

5.1.2内聚和耦合（内聚）

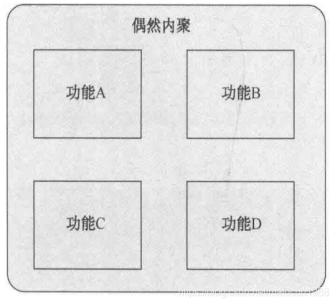
7种内聚的独立性

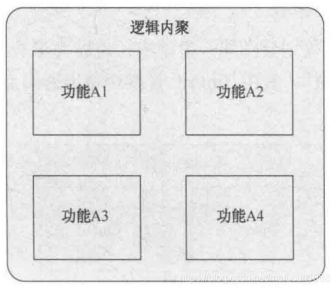
内聚： 模块内部各个元素彼此结合的紧密程度

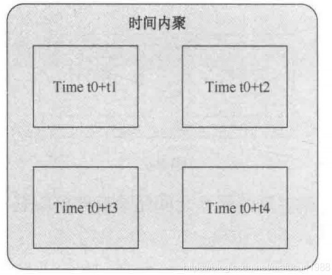


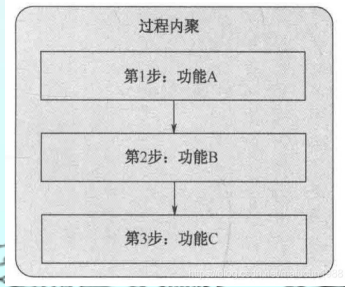
5.1.2内聚和耦合（内聚）

内聚名称	概念
偶然内聚	指一个模块内的各个部分是“任性”地组合到一起。
逻辑内聚	指一个模块内的几个组件仅仅因为“逻辑相似”而被放到了一起
时间内聚	多个组件除了要在程序执行到同一个时间点时做处理之外、没有其它关系。（比如系统初始化）
过程内聚	多个组件之间必须遵循一定的执行顺序才能完成一个完整功能



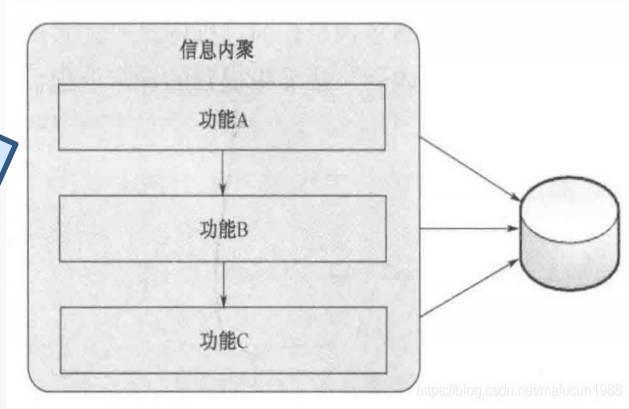


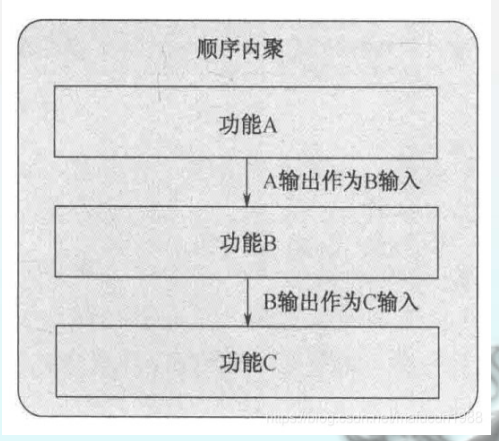


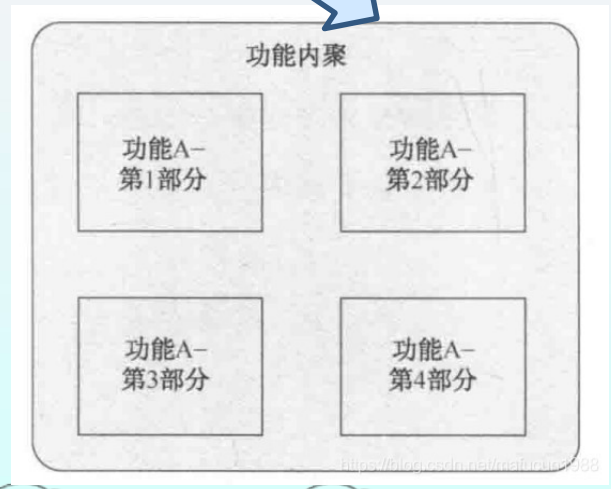


5.1.2内聚和耦合（内聚）

内聚名称	概念
通信内聚	所有组件都操作统一数据集，或生成统一数据集
顺序内聚	组件都相关同一功能，前一组件的输出就是下一组件的输入。
功能内聚	所有组件共同完成一个功能

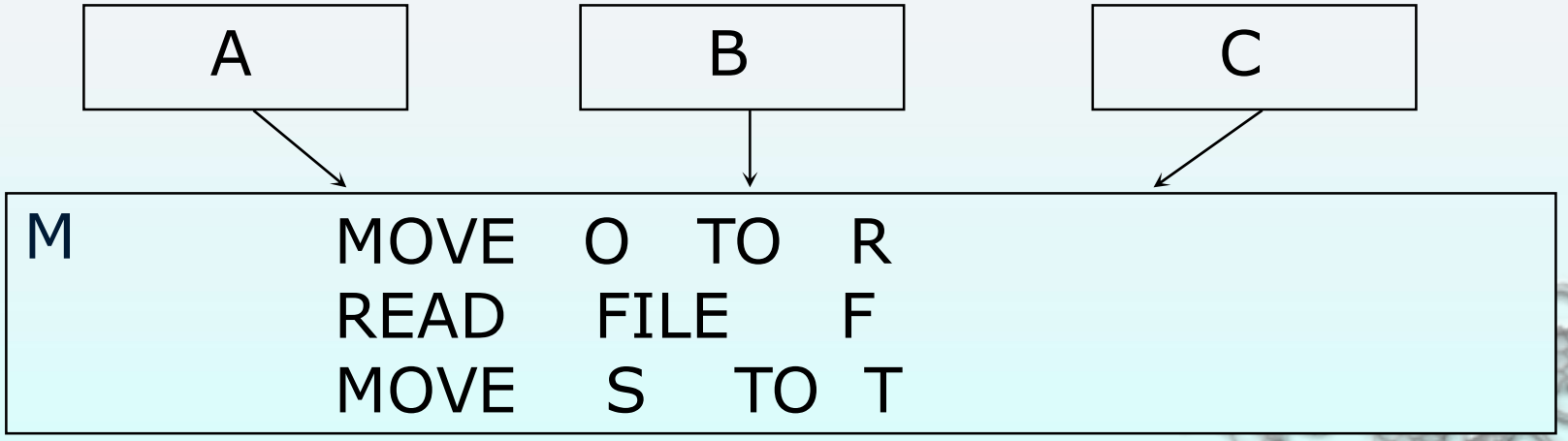






内聚和耦合（续）

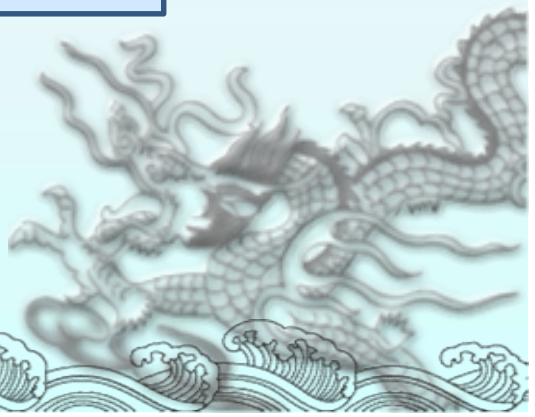
- 偶然内聚
  - 模块内各部分间无联系
  - 例子：模块M中的三个语句没有任何联系，但A、B和C模块中都用到这三条语句，因此将这三条语句合并成了模块M
  - 缺点：可理解性差， 可修改性差



内聚和耦合（续）

- 偶然内聚
  - 例子：这个方法需要获取名字，但方法里面做了两件事情，与方法的需要没有关联，它们之间也没有关联，与方法主题不符，即为偶然内聚。

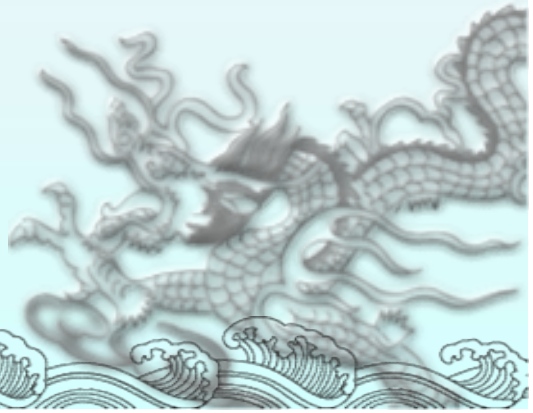
```
var getName = function(){
        var result = 1 + 2; //计算1+2
        console.log("hello"); //打印say hello
}
```



内聚和耦合（续）

- 偶然内聚
  - 例子：

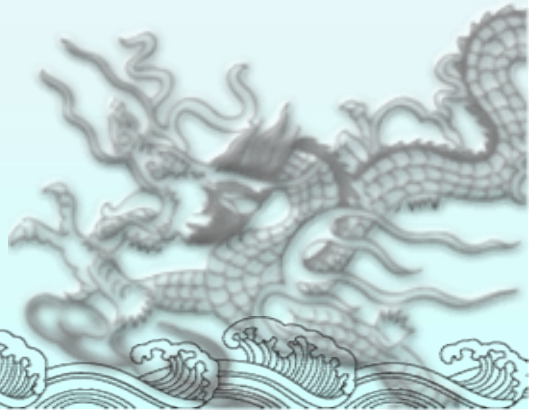
```
var theApple = {
        sex:"boy",
        font:"宋体"
}
```



内聚和耦合（续）

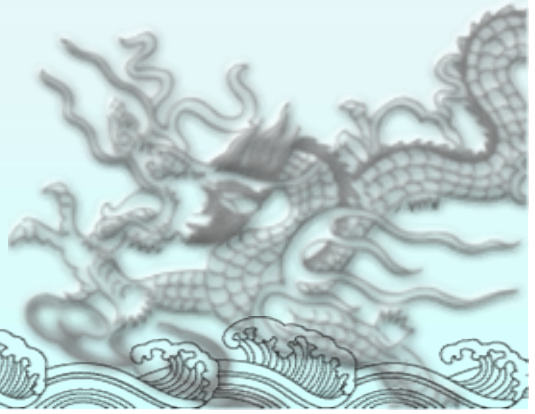
- 偶然内聚
  - 例子：这个苹果对象有两个属性，这两个属性与苹果毫无关系，两个属性之间也毫无关系，即为偶然内聚。

```
var theApple = {
        sex:"boy",
        font:"宋体"
}
```

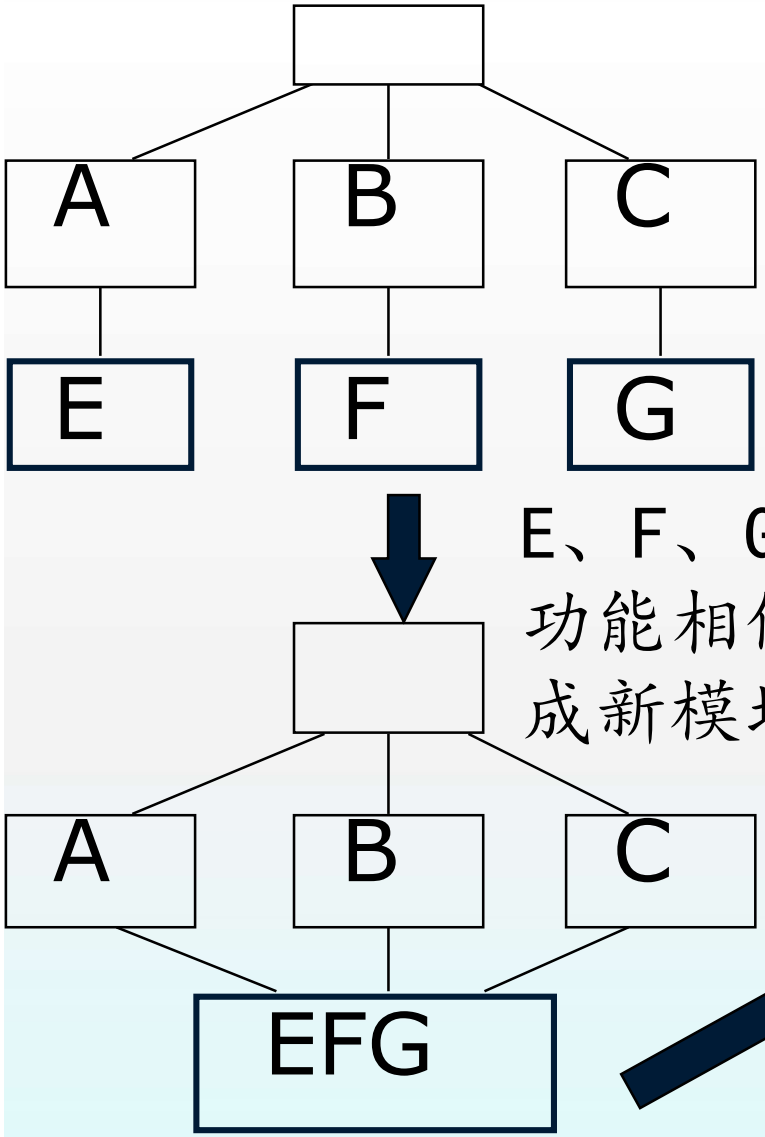


内聚和耦合（续）

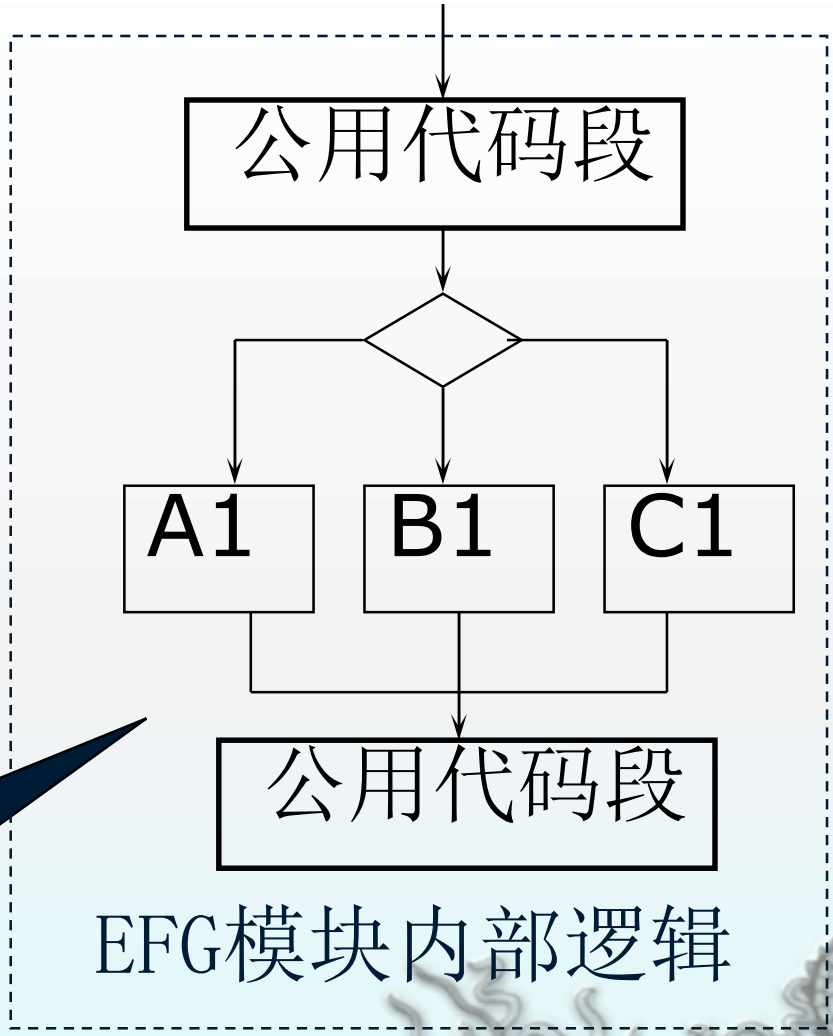
- 逻辑内聚
  - 把几种相关功能（逻辑上相似的功能）组合在一模块内
  - 每次调用由传给模块的参数确定执行哪种功能



逻辑内聚模块



E、F、G逻辑功能相似，组成新模块EFG



缺点：增强了耦合程度(控制耦合)
不易修改，效率低

内聚和耦合（续）

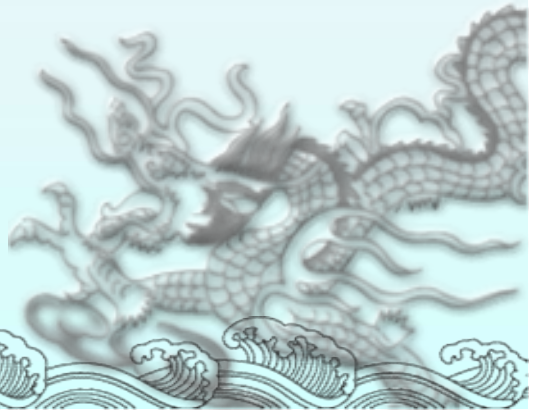
- 逻辑内聚
  - 例子：这是一个计算还款计划表的接口。入参中，LendApply 是借款申请；CalculateParam 是计算所需金额参数；CalcuateMethod 是计息方式——如等额本金、等额本息、先息后本，等等。

```
public interface RepayPlanCalculator{
  List<RepayPlan> calculate(LendApply apply,
                    CalculateParam param,
                    CalculateMethod calculateMethod);
}
```

内聚和耦合（续）

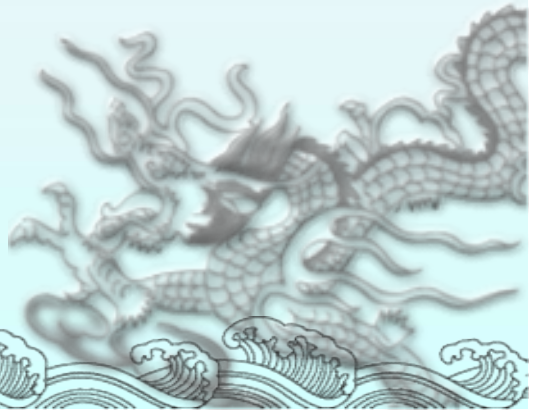
- 逻辑内聚
  - 例子：把接口内部的逻辑处理暴露到了接口之外。当逻辑发生变更时，可能会带来问题。

```
public interface RepayPlanCalculator{
  List<RepayPlan> calculate(LendApply apply,
                            CalculateParam param,
                            CalculateMethod calculateMethod);
}
```



内聚和耦合（续）

- 时间内聚
  - 模块完成的功能必须在同一时间内执行
  - 这些功能只因时间因素关联在一起。



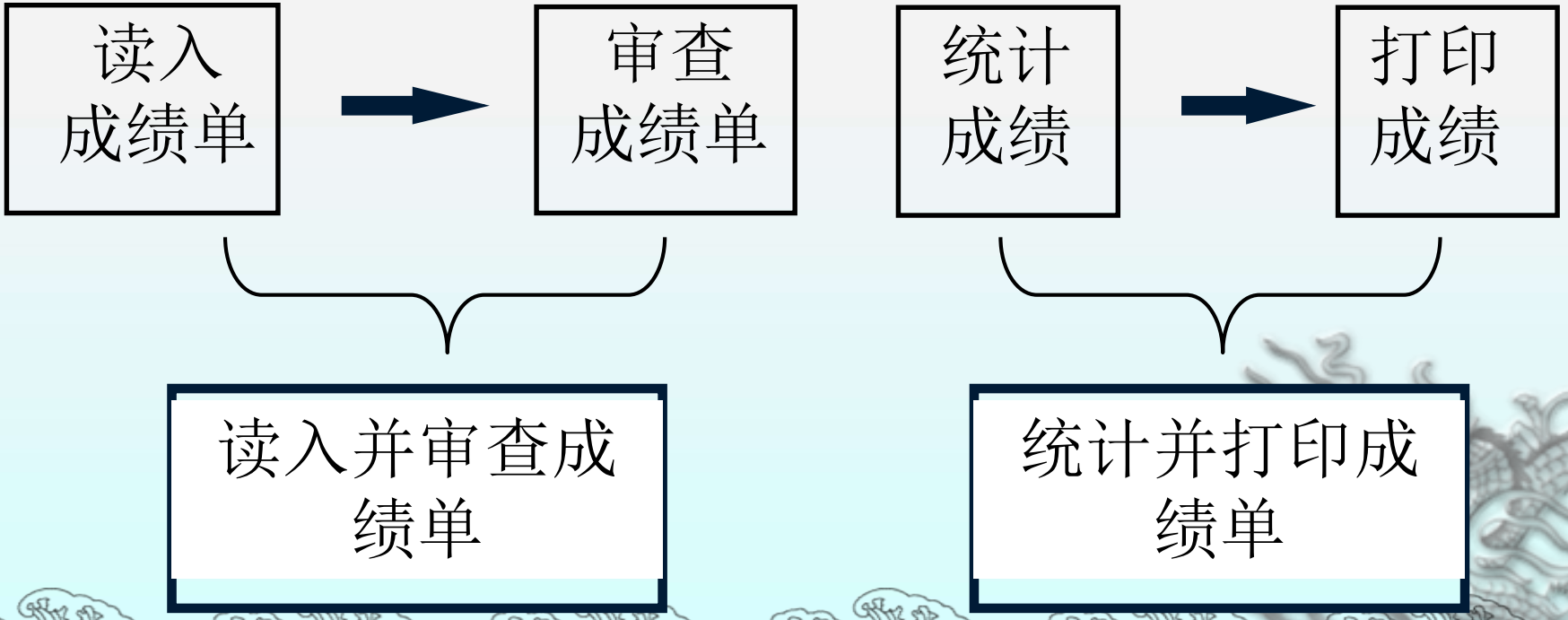
内聚和耦合（续）

- 时间内聚
  - 例子：

```
var system = {
        init(){
                this.logStartTime(); //记录系统启动时间日志
                this.initCache(); //初始化数据缓存
                this.checkData(); //检测启动数据
                this.startMessageService(); //启动消息服务
                //……
        },
        logStartTime(){}, //记录系统启动时间日志
        initCache(){},//初始化数据缓存
        checkData(){},//检测启动数据
        startMessageService(){} //启动消息服务
        //……
}
```

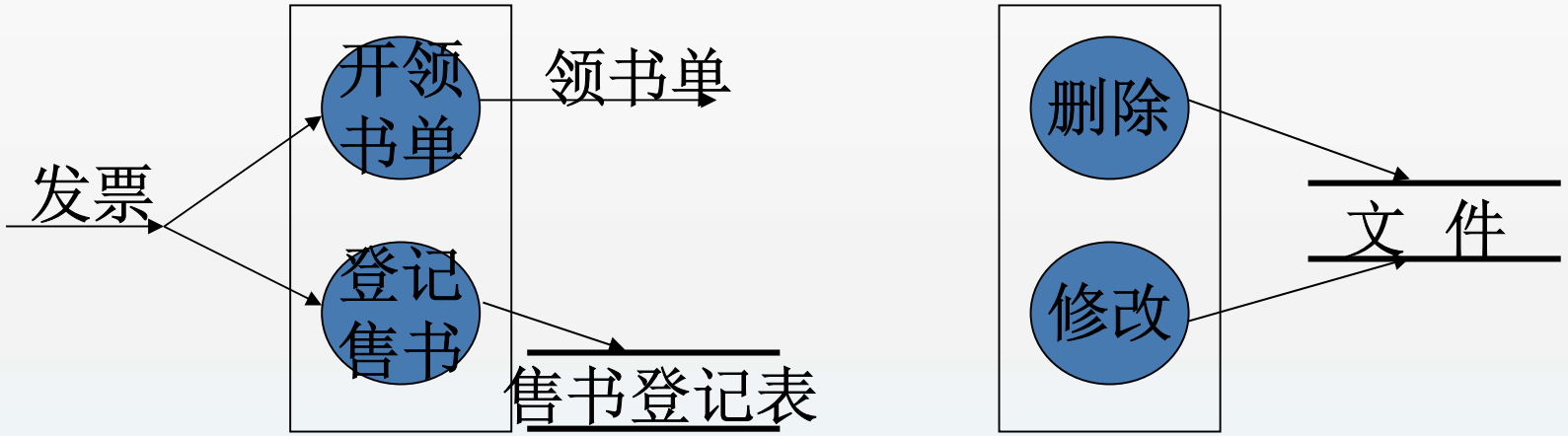
内聚和耦合（续）

- 过程内聚：
  - 模块内各处理成分相关
  - 必须以特定次序执行



内聚和耦合（续）

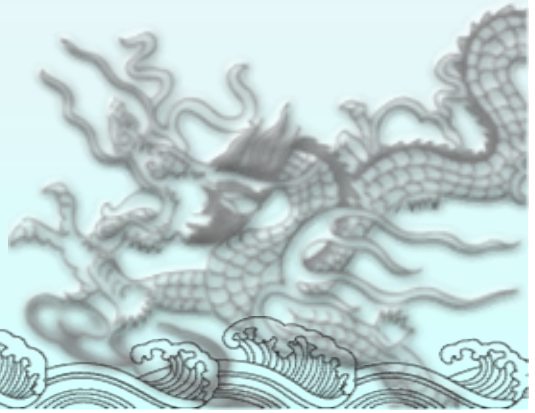
- 通信内聚：
  - 一个模块内各功能部分都针对相同输入/输出数据进行处理。




```
var userDAO = {
        getUserById(id){ …… },
        deleteById(id){ …… },
        insertOne(name, age, sex){ …… }
        updateNameById(id){ …… }
}
```

内聚和耦合（续）

- 顺序内聚
  - 模块中处理元素和同一个功能密切相关
  - 一个成分的输出作为另一个成分的输入
  - 处理元素必须是顺序执行的



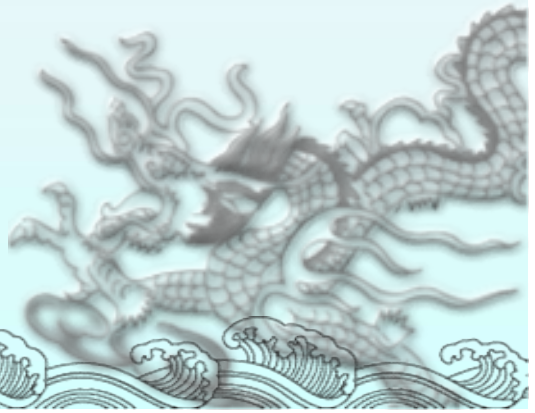
内聚和耦合（续）

- 顺序内聚
  - 例子：从 bankCardList.stream()开启一个Stream 之后，filter/map/map 每一步操作的输出都是下一个操作的输入，而且它们必须按顺序执行，这正是标准的顺序内聚。

```
List<BankCard> bankCardList = ...;
User u = ...;
String bankCardPhone =
 bankcCardList.stream()
 .filter(card->card.no().equals(u.getBankCardNo())))
 .map(BankCard::getPhone())
 .map(phone -> "*******" + phone.subString(phone.lengh()-4)))
 .orElse(StringUtils.EMPTY);
```

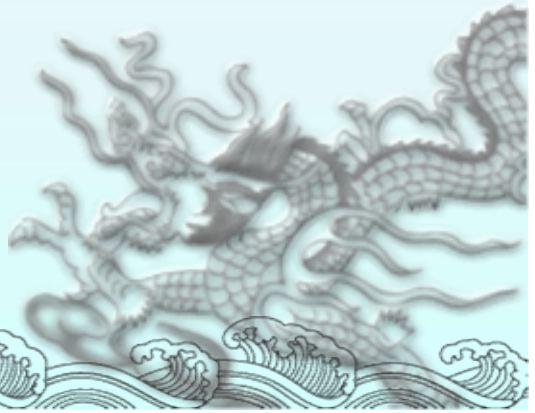
内聚和耦合（续）

- 顺序内聚
  - 例子：假设有一个按给出的生日计算雇员年龄、退休时间的子程序，如果它是利用所计算的年龄来确定雇员将要退休的时间，就是顺序内聚。



内聚和耦合（续）

- 功能内聚 ：一个模块中各个部分都是完成某一具体功能必不可少的组成部分，或者说该模块中所有部分都是为了完成一项具体功能而协同工作，紧密联系，不可分割的。



内聚和耦合（续）

- 功能内聚
- 例子：调用规则引擎的模块

```
public interface CallRuleService{
 RuleResult callRule(RuleData data);
}
class CallRuleServiceTemplate implements CallRuleService{
 public RuleResult callRule(RuleData data){
  validate(data);
  RuleRequest request = transToRequest(data);
  RuleResponse response = callRuleEngin(request);
  return transToResult(response);
 }
}
```

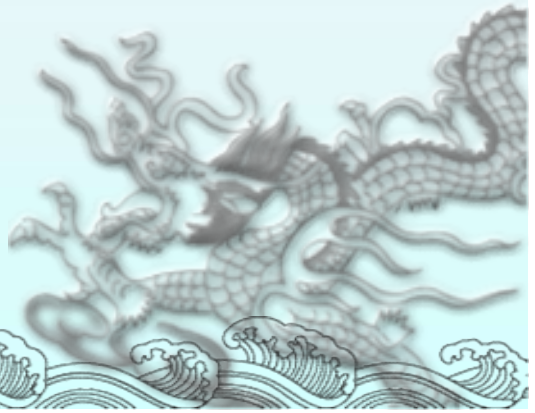
内聚和耦合（续）

- 功能内聚
- 例子：调用规则引擎的模块
- 这个模块包含校验、构建请求、调用规则引擎和解析结果这四个组件。无论是校验、构建请求、调用引擎还是解析结果，这个模块中所有的代码都是为了实现一个功能：调用规则引擎并解析结果。

```
public interface CallRuleService{
  RuleResult callRule(RuleData data);
}
class CallRuleServiceTemplate implements CallRuleService{
  public RuleResult callRule(RuleData data){
    validate(data);
    RuleRequest request = transToRequest(data);
    RuleResponse response = callRuleEngin(request);
    return transToResult(response);
  }
}
```

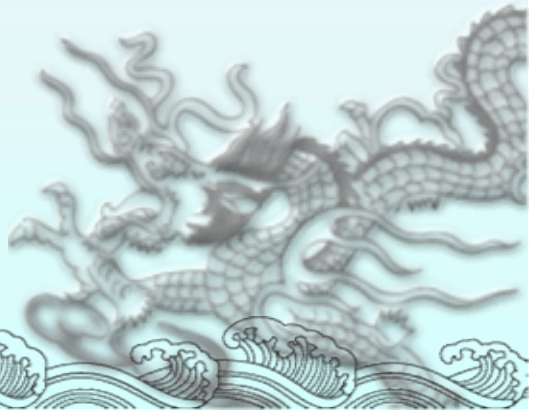
内聚总结：“一个模块，一个功能”

- 功能性内聚最强，与其他模块联系少，最优
- 其他的高内聚和中内聚模块也可以使用
- 低内聚模块尽量避免使用

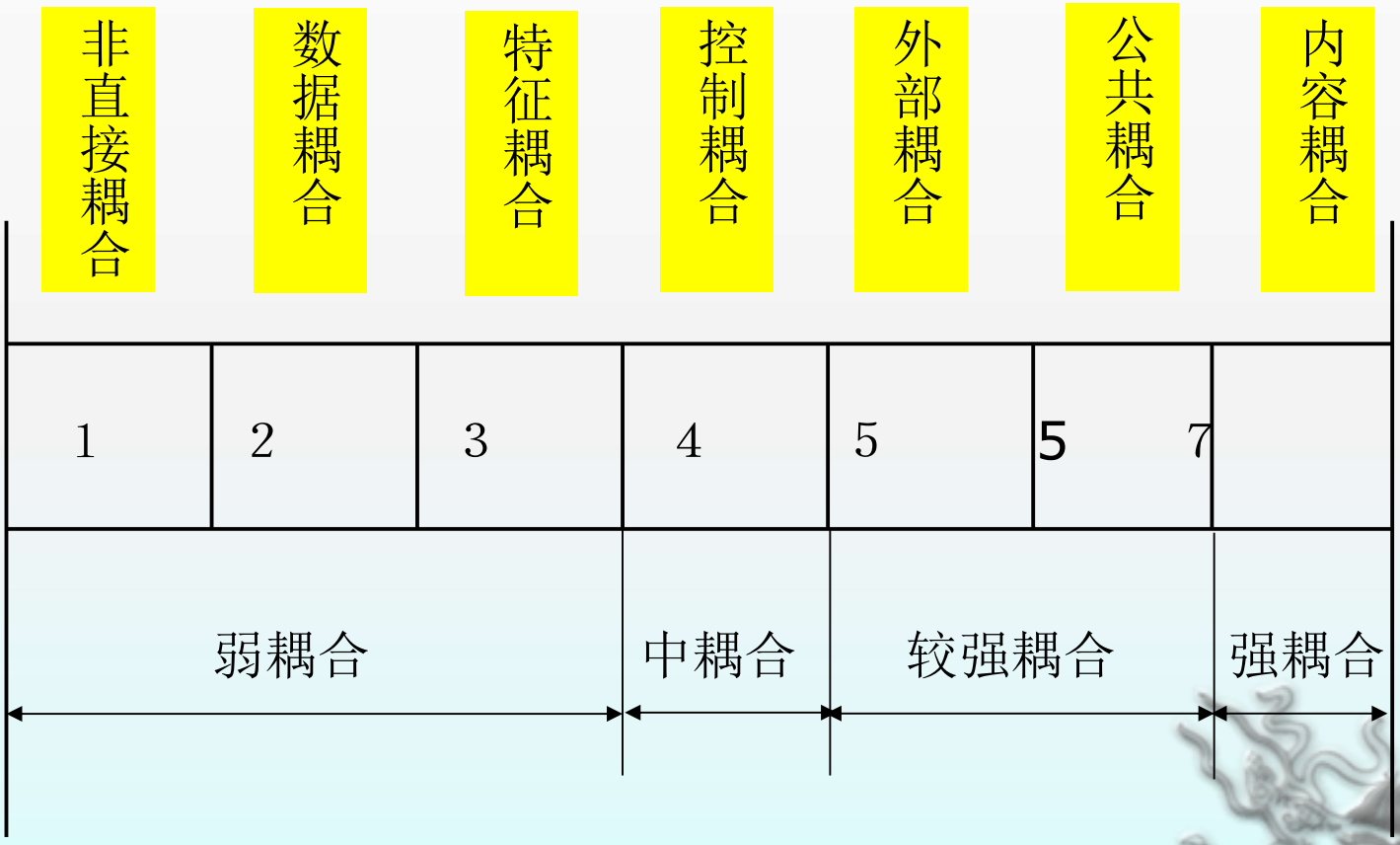


5.1.2 内聚和耦合（耦合）

- 模块间相互关联的程度取决于下面几点：
  - 一个模块对另一个模块的访问，比如模块A可能要调用模块B来完成一个功能。
  - 模块间传递的数据量。
  - 模块间接口的复杂程度。



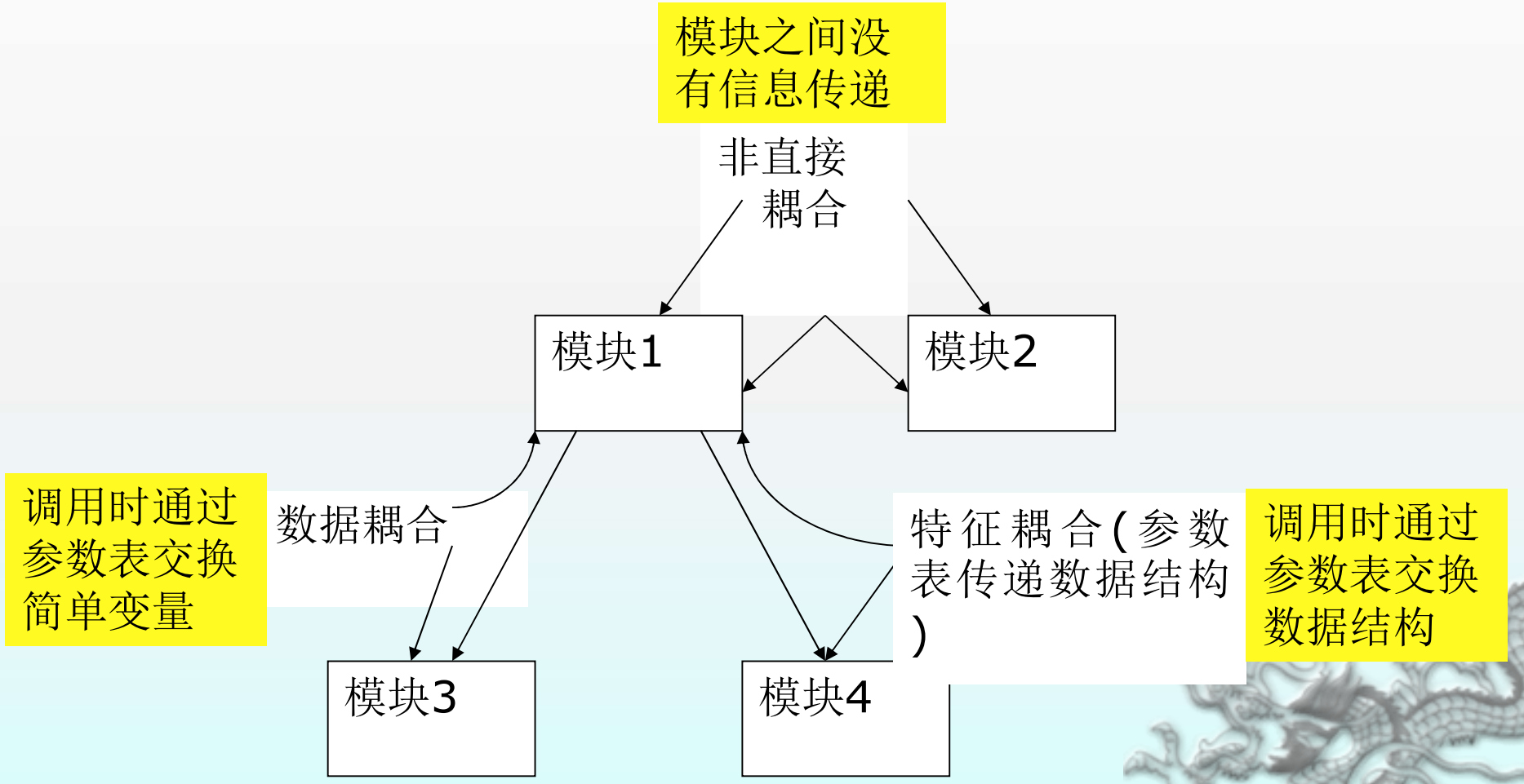
7种耦合的独立性



5.1.2内聚和耦合（耦合）

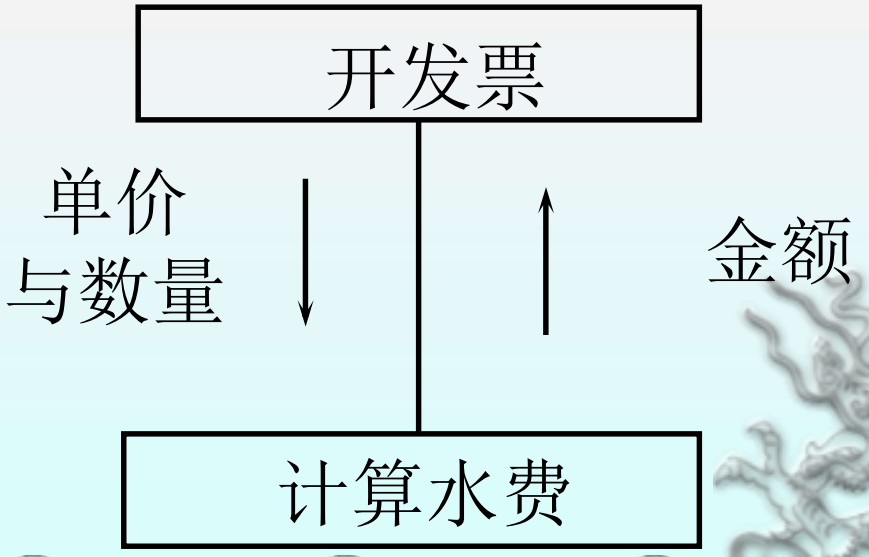
内聚名称	概念
内容耦合	指一个模块直接使用另一个模块的代码。这种耦合违反了信息隐藏这一基本的设计概念。
公共耦合	指多个模块访问同一个全局数据。当（某个模块或全局数据）发生变化时，这种耦合可能会导致不受控制的错误传播以及无法预见的副作用。
外部耦合	指两个模块共享一个外部强加的数据结构、通信协议或者设备接口。外部耦合基本上与外部工具和设备的通信有关。
控制耦合	指一个模块通过传入一个“做什么”的数据来控制另一个模块的流程。
特征耦合	是指多个模块共享一个数据结构、但是只使用了这个数据结构的一部分——可能各自使用了不同的部分。
数据耦合	指模块间通过传递数值来共享数据。传递的每个值都是基本数据，而且传递的值是就是要共享的值。
非直接耦合	指模块之间没有消息传递

弱耦合


模块之间没有信息传递
非直接耦合
模块1
模块2
调用时通过参数表交换简单变量
数据耦合
特征耦合(参数表传递数据结构)
调用时通过参数表交换数据结构
模块3
模块4

数据耦合

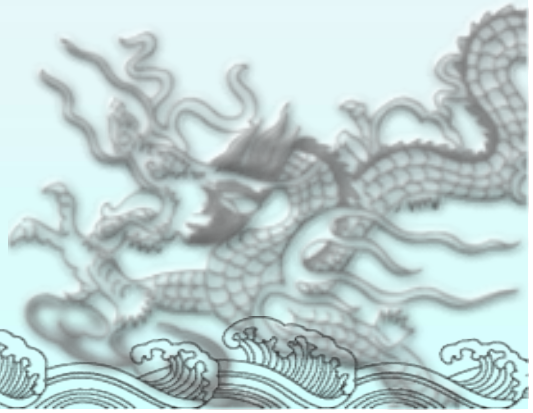
- 一模块调用另一模块时，被调用模块的输入、输出都是简单的数据(若干参数)
- 属松散耦合。



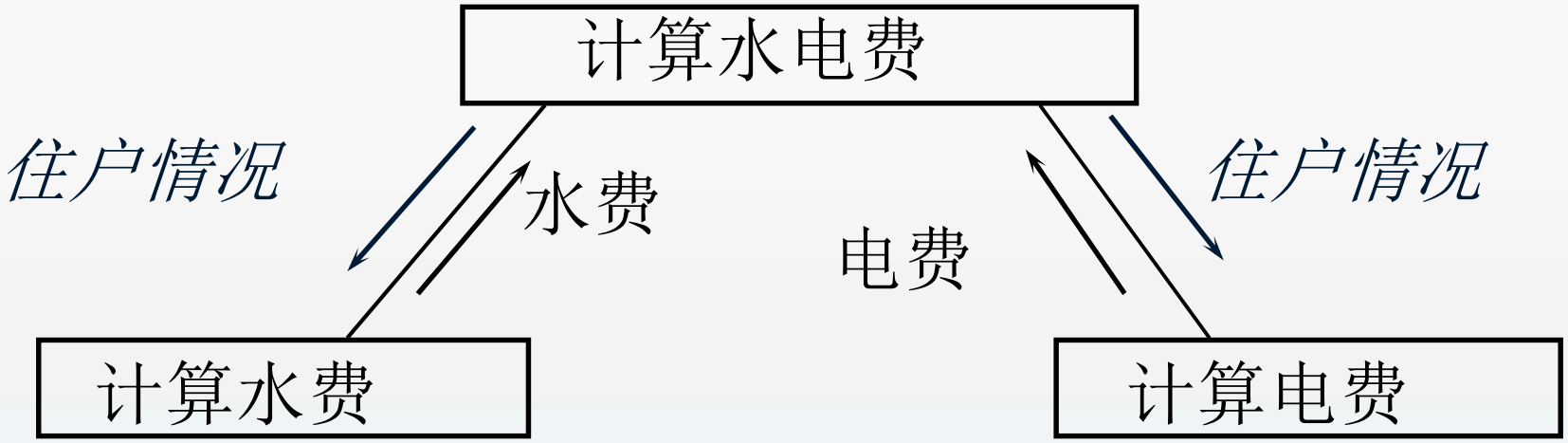
特征耦合(标记耦合)

- 两个模块通过传递数据结构(不是简单数据，而是记录、数组等)加以联系

- 都与一个数据结构有关系，则称这两个模块间存在特征偶合。

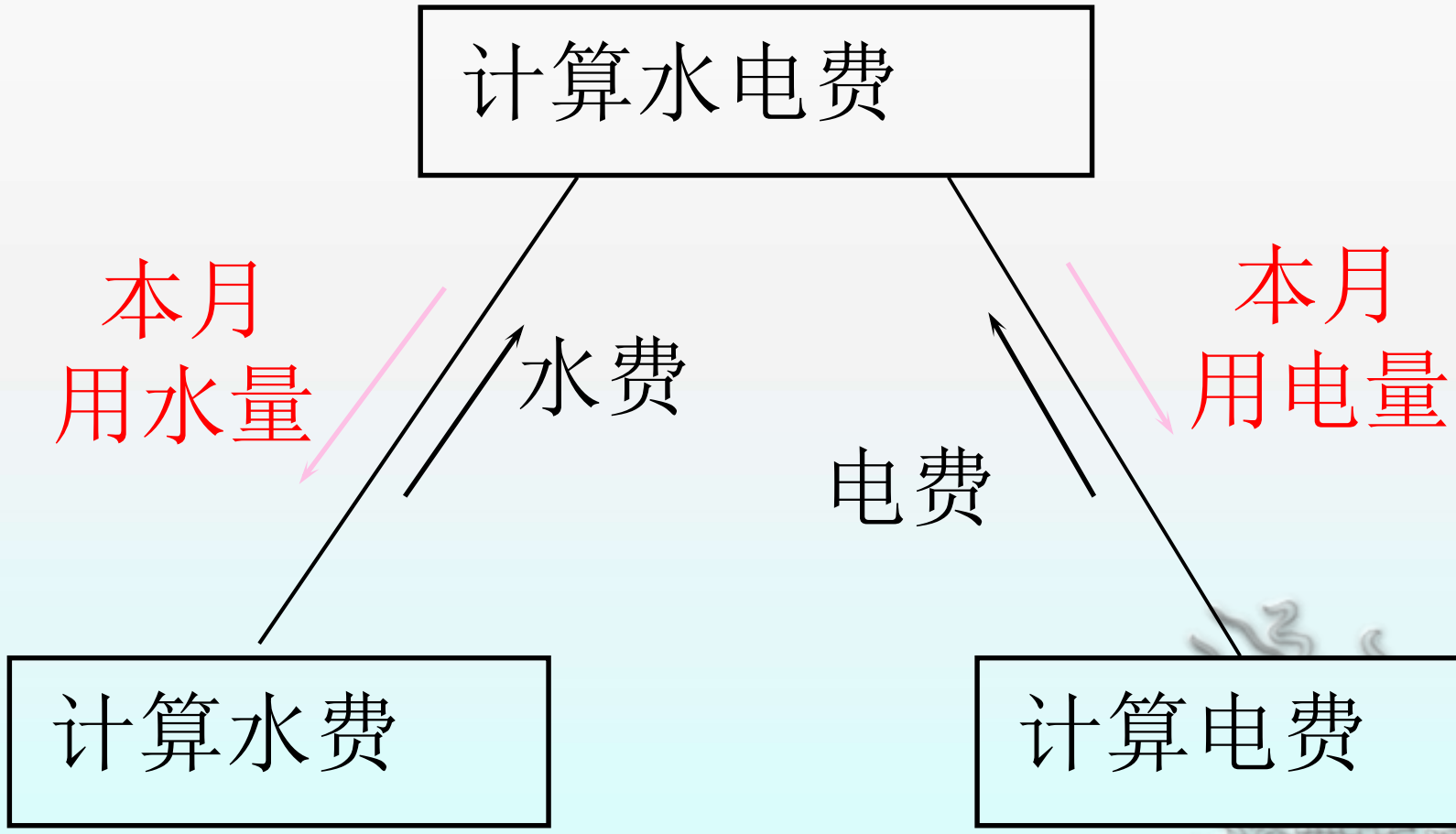


特征耦合举例



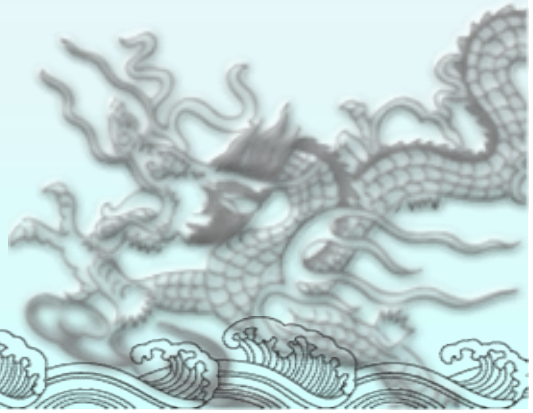
- “*住户情况*”是一个数据结构，图中模块都与此数据结构有关
- “计算水费”和“计算电费”本无关，由于引用了此数据结构产生依赖关系，它们之间也是特征偶合

将特征耦合改为数据耦合举例

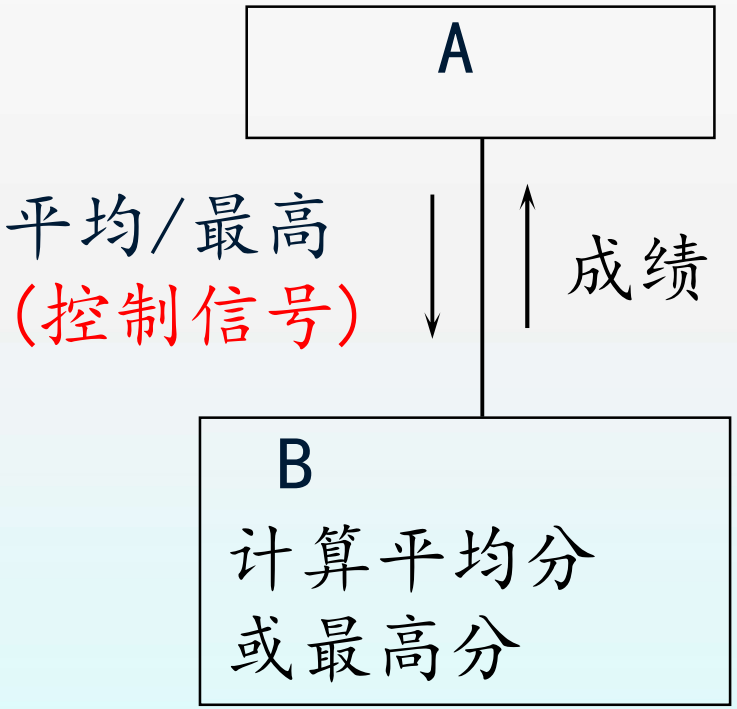


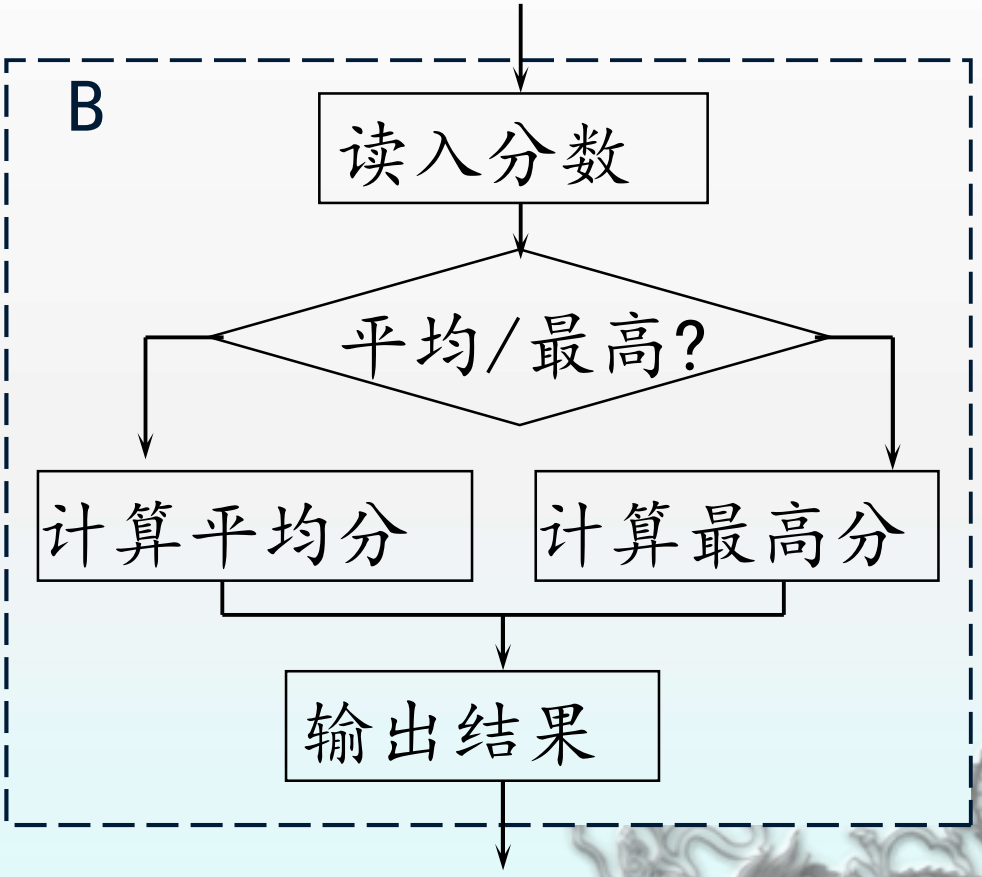
中耦合

- 控制耦合
  - 在模块间传递的信息是用作控制信号的开关值或标志量。
  - 控制模块必须知道被控制模块的内部逻辑，从而增强了模块间的相互依赖。



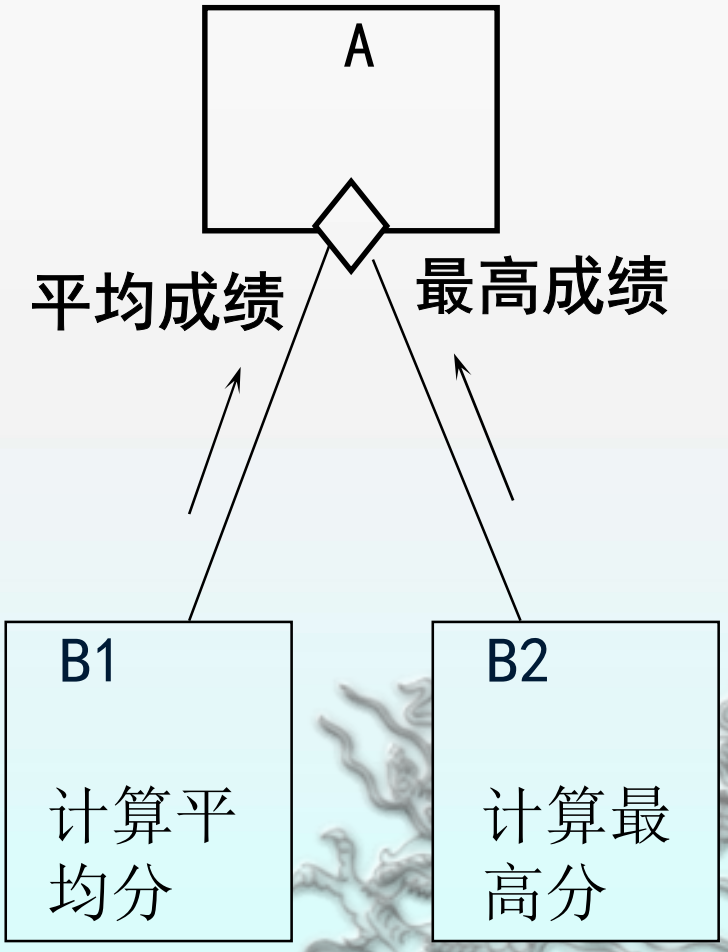
控制耦合举例





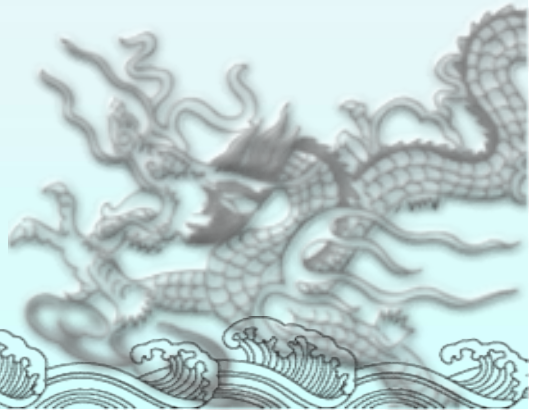
改控制耦合为数据耦合举例

- 去除模块间控制耦合的方法：
  - (1)将被调用模块内的判定上移到调用模块中进行
  - (2)被调用模块分解成若干单一功能模块

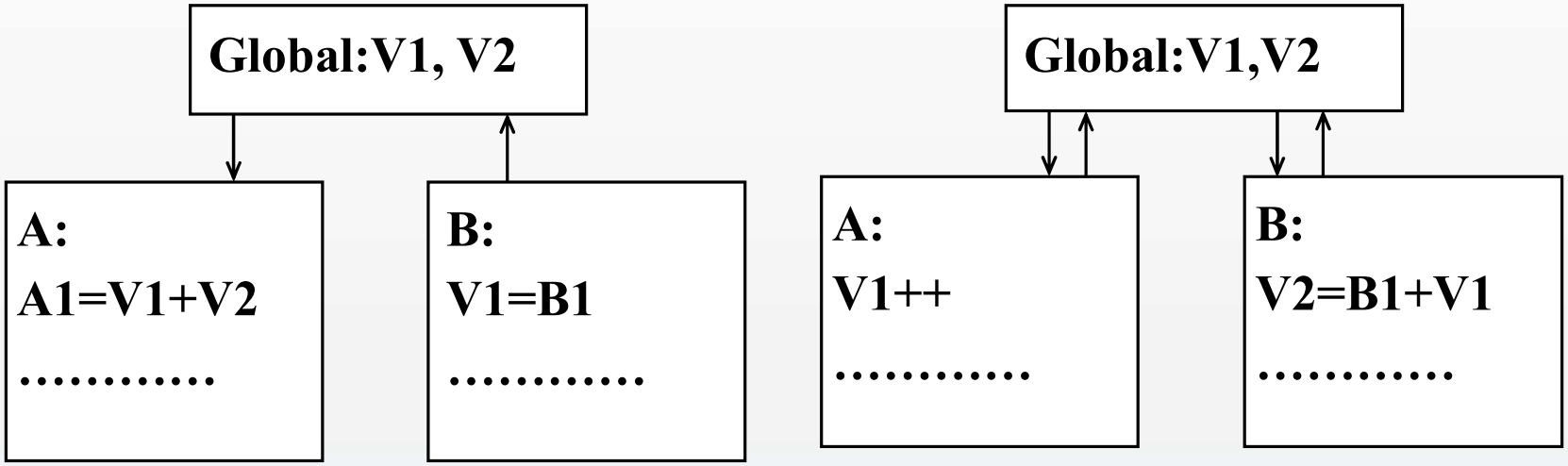


较强耦合

- 外部耦合
  - 允许一组模块访问同一个全局变量



公共耦合——多个模块都访问同一个公共数据环境，则称它们是公共耦合。

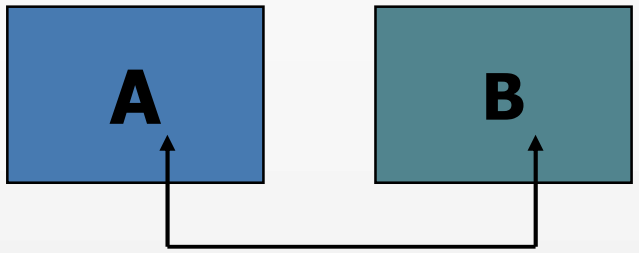


问题：公共部分的改动将影响所有调用它的模块；

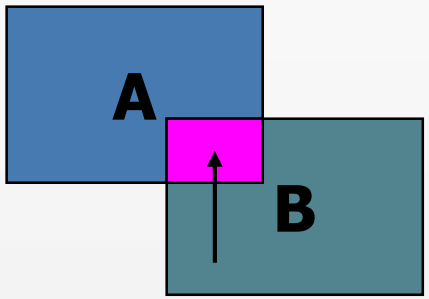
公共部分的数据存取无法控制；

复杂程度随耦合模块的个数增加而增加

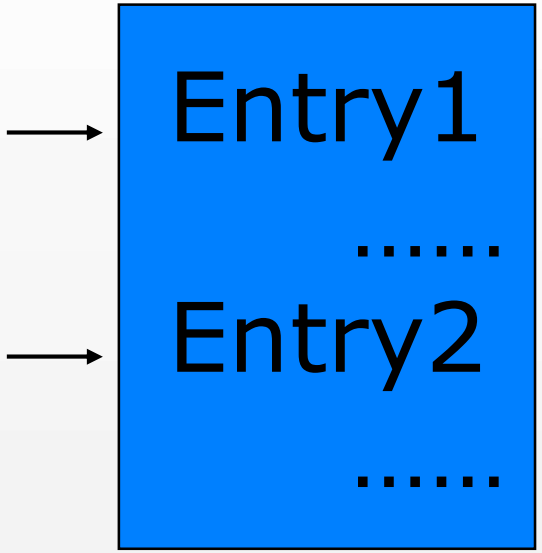
强耦合：内容耦合



一模块直接访问另
一模块的内部信息
(程序代码或数据)



模块代码重叠



多入口模块

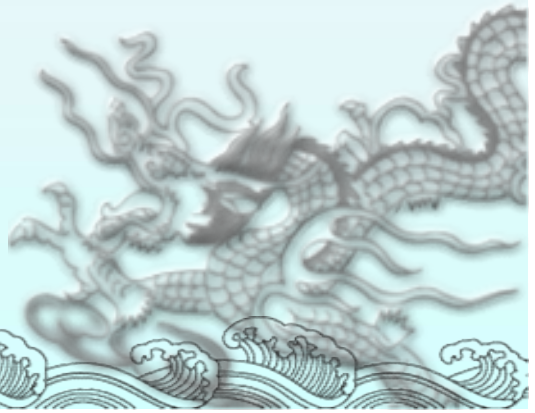
最不好的耦合形式 !!!

使用耦合的原则

对“耦合”的应用原则：不在于禁止耦合，而是充分了解各种耦合的特点与不足，并在需要时使用它们并能预见到可能产生的问题。

耦合是影响软件复杂程度的一个重要因素。应该采取下述设计原则：

尽量使用数据耦合，少用控制耦合，限制公共环境耦合的范围，完全不用内容耦合。



耦合例子

```
public class XxxService{
   private static final Bean BEAN = new Bean();
   static{
      // 初始化BEAN。Bean中所有get/set都是public的。BEAN.setA("a");
      BEAN.setB("b");
   }
   public List<Bean> queryBeanList(){
      // 先从数据库查一批数据
     List<Bean> list = ...;
     // 如果数据库么有数据，那么给个默认列表
     if(Collections.isEmpty(list)){
        list = Collections.singletonList(BEAN);
     }
     return list;
   }
}
// 使用上面这个方法的类
public class YyyService{
   public void doSomething(){
      List<Bean> beanList = xxxService.queryBeanList();
      // 其它逻辑，略
   }
}
public class ZzzService{
   public void querySomeList(){
      List<Bean> beanList = xxxService.queryBeanList();
      // 其他逻辑，略
   }
}
```

耦合例子

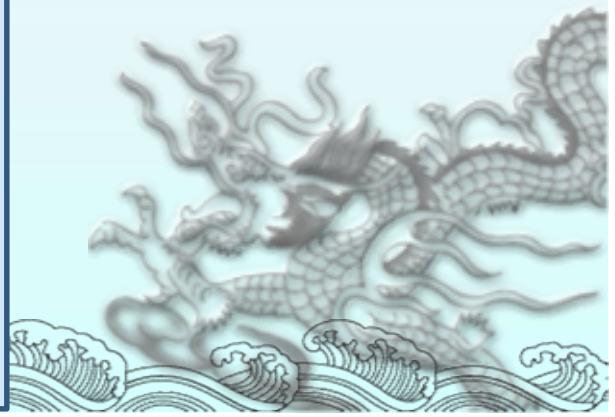
```
public class XxxService{
   private static final Bean BEAN = new Bean();
   static{
      // 初始化BEAN。Bean中所有get/set都是public的。BEAN.setA("a");
      BEAN.setB("b");
   }
   public List<Bean> queryBeanList(){
      // 先从数据库查一批数据
     List<Bean> list = ...;
     // 如果数据库没有数据，那么给个默认列表
     if(Collections.isEmpty(list)){
        list = Collections.singletonList(BEAN);
     }
     return list;
   }
}
// 使用上面这个方法的类
public class YyyService{
   public void doSomething(){
      List<Bean> beanList = xxxService.queryBeanList();
      // 其它逻辑，省略
   }
}
public class ZzzService{
   public void querySomeList(){
      List<Bean> beanList = xxxService.queryBeanList();
      // 其他逻辑，省略
   }
}
```

XxxService和**YyyService/ZzzService**（以及任何调用了**XxxService#queryBeanList()**方法的模块）之间，在**BEAN**这个全局变量上产生了公共耦合。

导致的问题

1）如果**XxxService**变更了**BEAN**中的数据——也就是变更了默认列表的数据——那么**YyyService/ZzzService**等模块就有可能受到不必要的牵连。

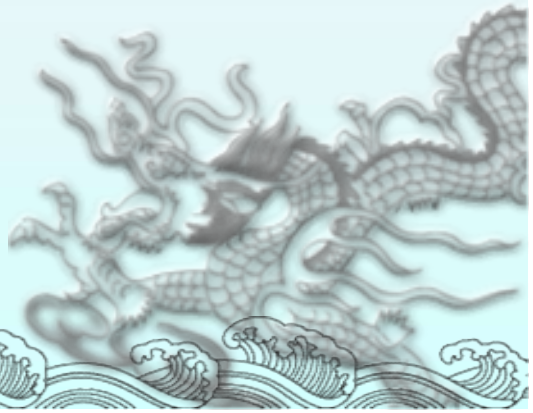
2）如果**YyyService**模块修改了**queryBeanList()**的返回数据，那就有可能修改**BEAN**中的数据，从而在悄无声息间改变了**queryBeanList()**的逻辑、并导致**ZzzService**模块出现莫名其妙的**bug**。



内聚与耦合练习

例1：请判断下面例子属于哪种内聚或耦合

```
int gcd()
{
        int a, b,t;
        scanf("%d %d", &a, &b);

        while (a%b!=0)
        {
                t = b;//存上一轮的除数
                b = a % b;//这一轮的余数做下一轮的除数
                a = t;//做下一轮的被除数
        }
        printf("%d", b);//最后剩下的除数就是答案
        return 0;
}
```



内聚与耦合练习

例1：请判断下面例子属于哪种内聚或耦合

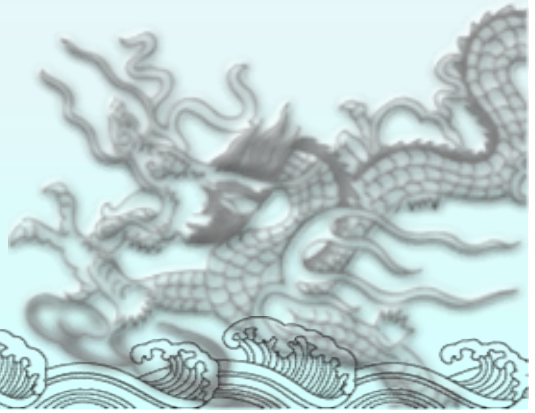
```
int gcd()
{
        int a, b,t;
        scanf("%d %d", &a, &b);

        while (a%b!=0)
        {
                t = b;//存上一轮的除数
                b = a % b;//这一轮的余数做下一轮的除数
                a = t;//做下一轮的被除数
        }
        printf("%d", b);//最后剩下的除数就是答案
        return 0;
}
```

最大公约数的计算方式，模块里面代码都为功能做出了贡献，为功能内聚。

内聚与耦合练习

例2：请判断下面例子属于哪种内聚或耦合

```
public void sample(String flag) {
        switch(flag) {
                case ON:
                        // ...
                        break;
                case OFF:
                        // ...
                        break;
                case CLOSE:
                        // ...
                        break;
        }
}
```



内聚与耦合练习

例2：请判断下面例子属于哪种内聚或耦合

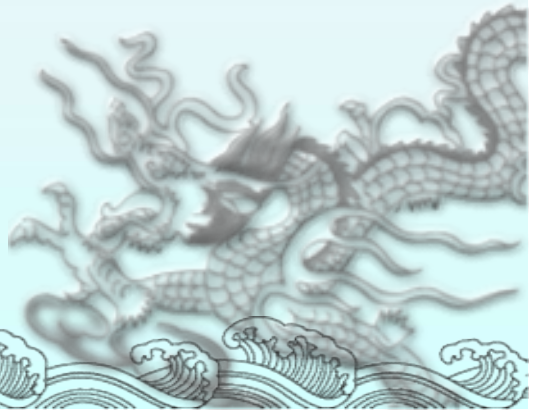
```
public void sample(String flag) {
        switch(flag) {
                case ON:
                        // ...
                        break;
                case OFF:
                        // ...
                        break;
                case CLOSE:
                        // ...
                        break;
        }
}
```

具有相似的逻辑，放在了一起，为逻辑内聚。

内聚与耦合练习

例3：请判断下面例子属于哪种内聚或耦合

```
void validate_checkout_request(input_form i) {

        if(!(i.name.size()>4 && i.name.size()<20)) {
                error_message("Invalid name");
        }

        if(!(i.date.month>=1 && i.date.month<=12)) {
                error_message("Invalid month");
        }
}
```

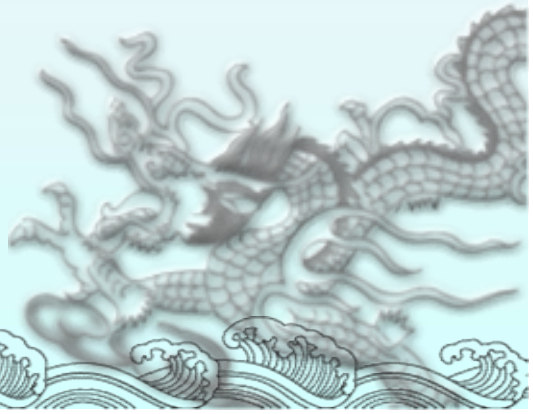


内聚与耦合练习

例3：请判断下面例子属于哪种内聚或耦合

```
void validate_checkout_request(input_form i) {

        if(!(i.name.size()>4 && i.name.size()<20)) {
                error_message("Invalid name");
        }

        if(!(i.date.month>=1 && i.date.month<=12)) {
                error_message("Invalid month");
        }
}
```

为什么不是逻辑内聚？



内聚与耦合练习

例3：请判断下面例子属于哪种内聚或耦合

```
void validate_checkout_request(input_form i) {

        if(!(i.name.size()>4 && i.name.size()<20)) {
                error_message("Invalid name");
        }

        if(!(i.date.month>=1 && i.date.month<=12)) {
                error_message("Invalid month");
        }
}
```

答案之一：多次调用了形参 i 的不同属性，模块执行的操作是在相同的数据上进行的，故为通信内聚。

内聚与耦合练习

例3：请判断下面例子属于哪种内聚或耦合

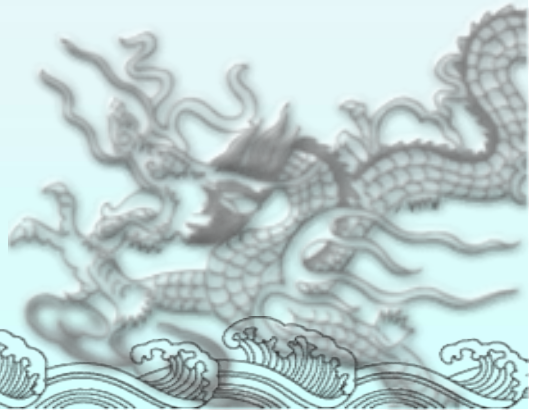
```
void validate_checkout_request(input_form i) {

        if(!(i.name.size()>4 && i.name.size()<20)) {
                error_message("Invalid name");
        }

        if(!(i.date.month>=1 && i.date.month<=12)) {
                error_message("Invalid month");
        }
}
```

答案之二：为了完成单一的功能，故也可以认为是功能内聚。

内聚与耦合练习

例4：请判断下面例子属于哪种内聚或耦合

```
Class A {
        Private:
                FinancialReport fr;
                WeatherDate wd;
                int totalcount;
        Public:
                void init();
}

void init() { // 初始化模块
        // 初始化财务报告
        fr = new (FinancialReport);
        fr.setRatio(5);
        fr.setYear("2010");
        // 初始化当前天气
        w = new(WeatherData);
        w.setCity("NanJing");
        w.setCode("210093");
        // 初始化计算器
        totalCount = 0;
}
```



内聚与耦合练习

例4：请判断下面例子属于哪种内聚或耦合

```
Class A {
        Private:
                FinancialReport fr;
                WeatherDate wd;
                int totalcount;
        Public:
                void init();
}

void init() { // 初始化模块
        // 初始化财务报告
        fr = new (FinancialReport);
        fr.setRatio(5);
        fr.setYear("2010");
        // 初始化当前天气
        w = new(WeatherData);
        w.setCity("NanJing");
        w.setCode("210093");
        // 初始化计算器
        totalCount = 0;
}
```

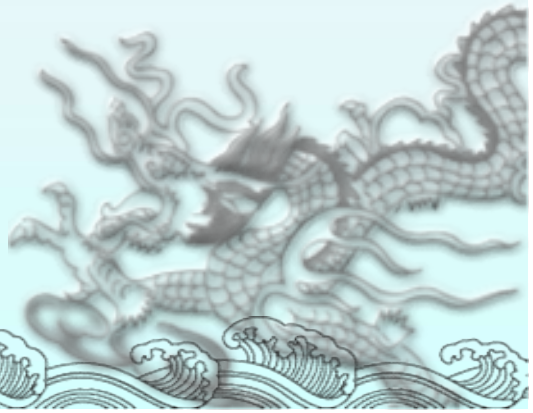
答案：初始化方法都在同一时间段内发生，为时间内聚。

内聚与耦合练习

例4：请判断下面例子属于哪种内聚或耦合

```
Class A {
        Private:
                FinancialReport fr;
                WeatherDate wd;
                int totalcount;
        Public:
                void init();
}

void init() { // 初始化模块
        // 初始化财务报告
        fr = new (FinancialReport);
        fr.setRatio(5);
        fr.setYear("2010");
        // 初始化当前天气
        w = new(WeatherData);
        w.setCity("NanJing");
        w.setCode("210093");
        // 初始化计算器
        totalCount = 0;
}
```

代码维护起来困难，怎么改进？



内聚与耦合练习

例4：请判断下面例子属于哪种内聚或耦合

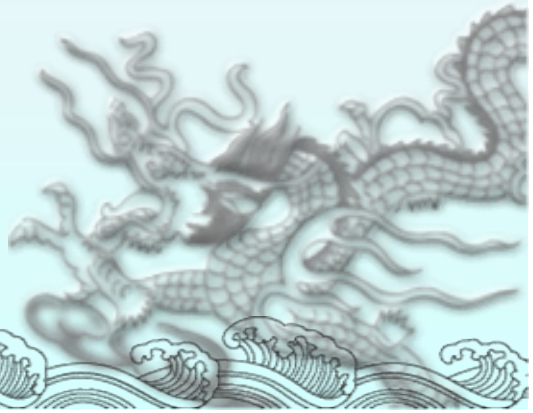
```
public class A {
        Private:
                FinancialReport fr;
                WeatherData wd;
                int totalcount;
        Public:
                void initFianceReport();
                void init WeatherDate();
                void initToalcount();
}
```

```
void initFinanceReport() {
        // 初始化财务报告
        fr = new(FinanceReport);
        fr.setRatio(5);
        fr.setYear("2010");
}

void initWeatherData() {
        // 初始化当前天气
        w = new(WeatherData);
        w.setCity("NanJing");
        w.setCode("210093");
}

void initTotalCount() {
        // 初始化计数器
        totalCount = 0;
}
```

内聚与耦合练习

例5：请判断下面例子属于哪种内聚或耦合

```
public class Rous {
        public static int findPattern(String text, String pattern)
{
                // …
        }
        public static int average(Vector numbers) {
                // …
        }
        public static OutputStream openFile(String fileName) {
                // …
        }
}
```



内聚与耦合练习

例5：请判断下面例子属于哪种内聚或耦合

```
public class Rous {
        public static int findPattern(String text, String pattern) {
                // ...
        }
        public static int average(Vector numbers) {
                // ...
        }
        public static OutputStream openFile(String fileName) {
                // ...
        }
}
```

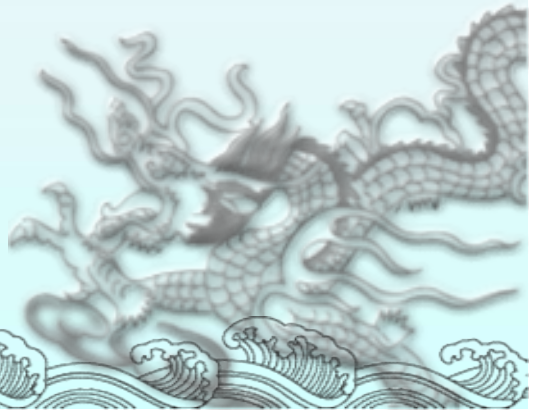
答案：模块中查询、计算、文件操作看上去不相关，为偶然内聚。

内聚与耦合练习

例5：请判断下面例子属于哪种内聚或耦合

```
public class Rous {
        public static int findPattern(String text, String pattern) {
                // ...
        }
        public static int average(Vector numbers) {
                // ...
        }
        public static OutputStream openFile(String fileName) {
                // ...
        }
}
```

怎么改进？



内聚与耦合练习

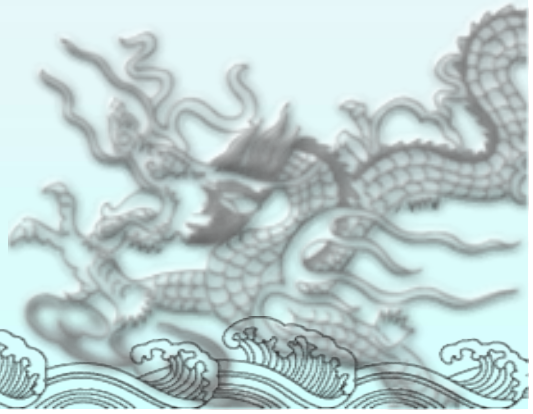
例5：请判断下面例子属于哪种内聚或耦合

```
public class RousFind {
        public static int findPattern(String text, String pattern) {
                // ...
        }
}

public class RousAve {
        public static int averager(Vector numbers) {
                // ...
        }
}

public class RousOpen {
        public static OutputStream openFile(String fileName) {
                // ...
        }
}
```

内聚与耦合练习

例6：请判断下面例子属于哪种内聚或耦合

```
public class foo {
        private String name;
        private int size;
        public void foo() { // 构造函数
                this.name = "Not Set";
                this.size = 12;
        }
        public void ~foo() { // 析构函数
                delete[] name;
                delete size;
        }
}
```

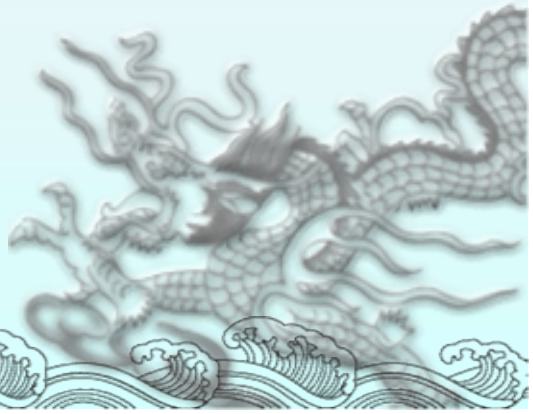


内聚与耦合练习

例6：请判断下面例子属于哪种内聚或耦合

```
public class foo {
        private String name;
        private int size;
        public void foo() { // 构造函数
                this.name = "Not Set";
                this.size = 12;
        }
        public void ~foo() { // 析构函数
                delete[] name;
                delete size;
        }
}
```

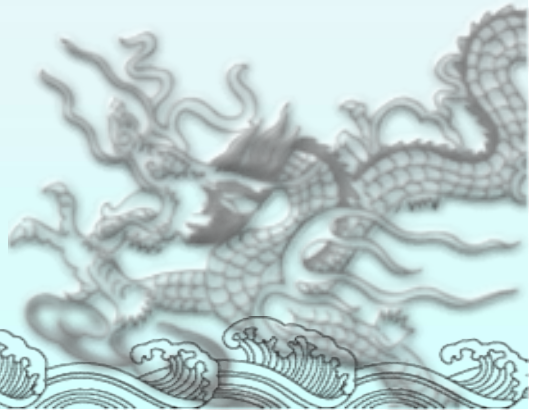
答案：构造函数和析构函数通常是时间内聚。



内聚与耦合练习

例7：请判断下面例子属于哪种内聚或耦合

```
public class Vector3D {
        public int x,y,z;
        //……
}

public class Arch {
        private Vector3D baseline;
        //…
        void slant(int newY) {
                baseline.x = 10;
                baseline.y = 13;
        }
}
```



内聚与耦合练习

例7：请判断下面例子属于哪种内聚或耦合

```
public class Vector3D {
        public int x,y,z;
        //……
}

public class Arch {
        private Vector3D baseline;
        //…
        void slant(int newY) {
                baseline.x = 10;
                baseline.y = 13;
        }
}
```

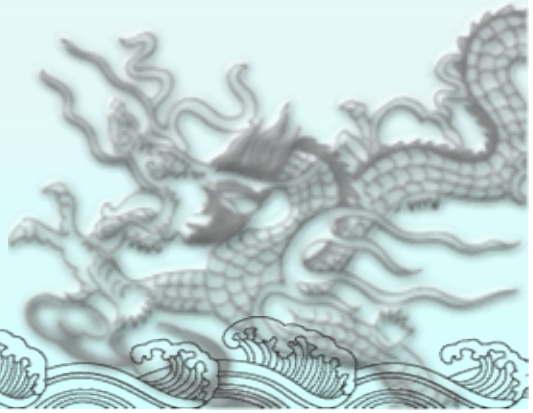
答案：**Arch**直接使用了**Vector3D** 的变量，属于内容耦合。

内聚与耦合练习

例7：请判断下面例子属于哪种内聚或耦合

```
public class Vector3D {
        public int x,y,z;
        //……
}

public class Arch {
        private Vector3D baseline;
        //…
        void slant(int newY) {
                baseline.x = 10;
                baseline.y = 13;
        }
}
```

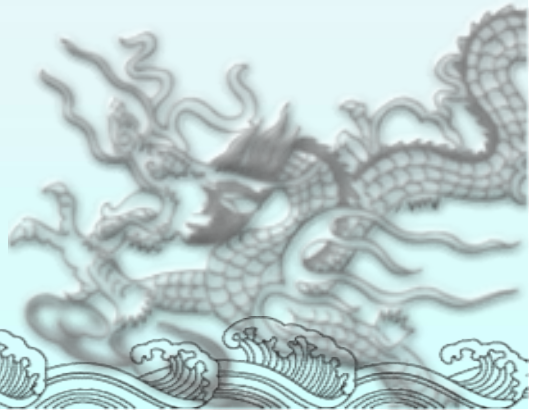
怎么改进？



内聚与耦合练习

例7：请判断下面例子属于哪种内聚或耦合

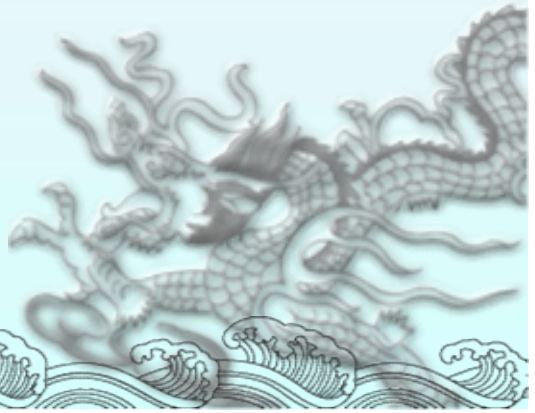
```
public class Vertor3D {
        private int x, y, z;
        // get/set 方法
        …
}

public class Arch {
        private Vertor3D baseline;
        //…
        void slant(int newY) {
                baseline.setX(10);
                baseline.setY(13);
        }
}
```



内聚与耦合练习

例8：请判断下面例子属于哪种内聚或耦合

```
public routineX(String command) {
        if(command.equals("drawCircle")) {
                drawCircle();
        }
        else {
                drawRectangle();
        }
}
```



内聚与耦合练习

例8：请判断下面例子属于哪种内聚或耦合

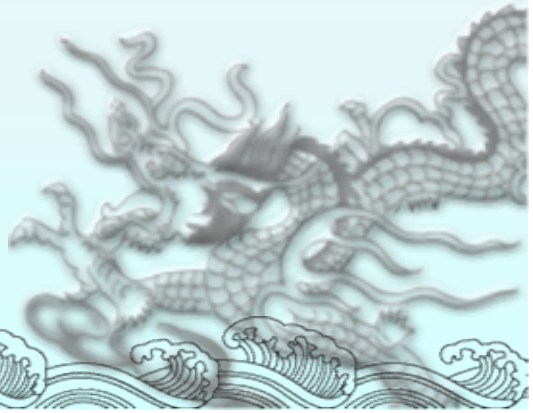
```
public routineX(String command) {
        if(command.equals("drawCircle")) {
                drawCircle();
        }
        else {
                drawRectangle();
        }
}
```

答案：存在某个模块给 **routineX** 模块传递 **command**，两个模块都需要知道：动词“画”+名词“圆”这个逻辑，才可以完成相应的操作，属于控制耦合。

内聚与耦合练习

例9：请判断下面例子属于哪种内聚或耦合

```
int x;

public class myValue {
        public void addValue(int a) {
                x = x + a;
        }
        public void subtractValue(int a) {
                x = x - a;
        }
}
```



内聚与耦合练习

例9：请判断下面例子属于哪种内聚或耦合

```
int x;

public class myValue {
        public void addValue(int a) {
                x = x + a;
        }
        public void subtractValue(int a) {
                x = x - a;
        }
}
```

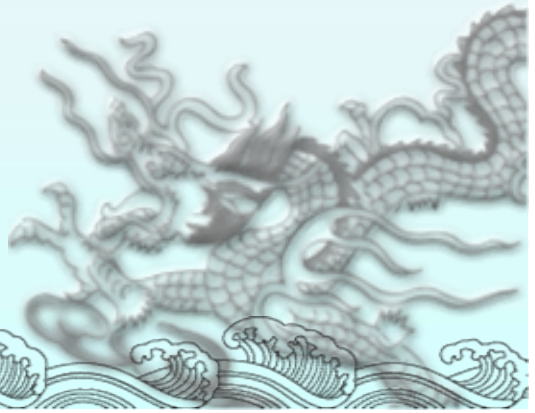
答案：多个模块都访问同一个公共数据环境，属于公共耦合。

内聚与耦合练习

例9：请判断下面例子属于哪种内聚或耦合

```
int x;

public class myValue {
        public void addValue(int a) {
                x = x + a;
        }
        public void subtractValue(int a) {
                x = x - a;
        }
}
```

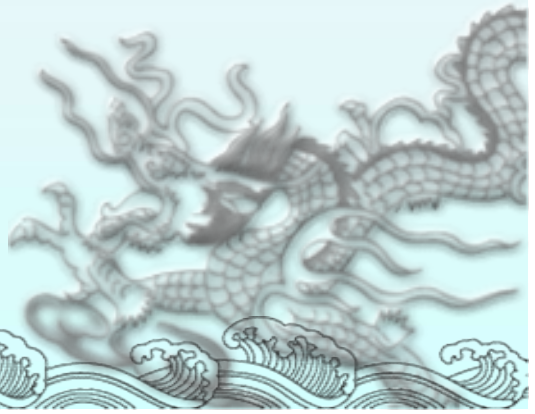
怎么改进？



内聚与耦合练习

例9：请判断下面例子属于哪种内聚或耦合

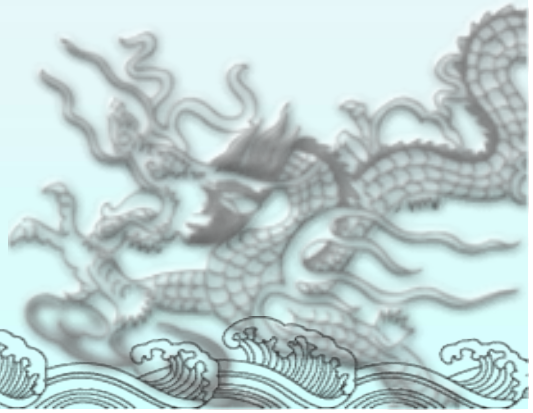
```
public class myValue {
        public int addValue(int a, int x) {
                return x + a;
        }
        public int subtractValue(int a, int x) {
                return x - a;
        }
}
```



内聚与耦合练习

例10：请判断下面例子属于哪种内聚或耦合

```
void validate_checkout_request(input_form i) {
        if(!valid_string(i.name)) {
                i.string = "Invalid name";
                error_message();
        }
        if(!valid_string(i.book)) {
                i.string = "Invalid book name";
                error_message();
        }
        valid_month(i.date);
}
void valid_month(Date d) {
        if(d.month < 1) {
                d.month = 1;
        }
        if(d.month > 12) {
                d.month = 12;
        }
        return 1;
}
```



内聚与耦合练习

例10：请判断下面例子属于哪种内聚或耦合

```
void validate_checkout_request(input_form i) {
        if(!valid_string(i.name)) {
                i.string = "Invalid name";
                error_message();
        }
        if(!valid_string(i.book)) {
                i.string = "Invalid book name";
                error_message();
        }
        valid_month(i.date);
}
void valid_month(Date d) {
        if(d.month < 1) {
                d.month = 1;
        }
        if(d.month > 12) {
                d.month = 12;
        }
        return 1;
}
```

答案：

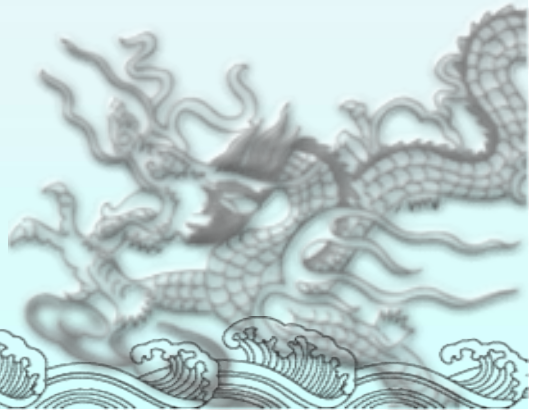
validate_checkout_request 多次调用了形参 **i** 的不同属性，模块执行的操作是在相同的数据上进行的，故为通信内聚。

内聚与耦合练习

例10：请判断下面例子属于哪种内聚或耦合

```
void validate_checkout_request(input_form i) {
        if(!valid_string(i.name)) {
                i.string = "Invalid name";
                error_message();
        }
        if(!valid_string(i.book)) {
                i.string = "Invalid book name";
                error_message();
        }
        valid_month(i.date);
}
void valid_month(Date d) {
        if(d.month < 1) {
                d.month = 1;
        }
        if(d.month > 12) {
                d.month = 12;
        }
        return 1;
}
```

是否还有其他的内聚或耦合？



内聚与耦合练习

例10：请判断下面例子属于哪种内聚或耦合

```
void validate_checkout_request(input_form i) {
        if(!valid_string(i.name)) {
                i.string = "Invalid name";
                error_message();
        }
        if(!valid_string(i.book)) {
                i.string = "Invalid book name";
                error_message();
        }
        valid_month(i.date);
}
void valid_month(Date d) {
        if(d.month < 1) {
                d.month = 1;
        }
        if(d.month > 12) {
                d.month = 12;
        }
        return 1;
}
```

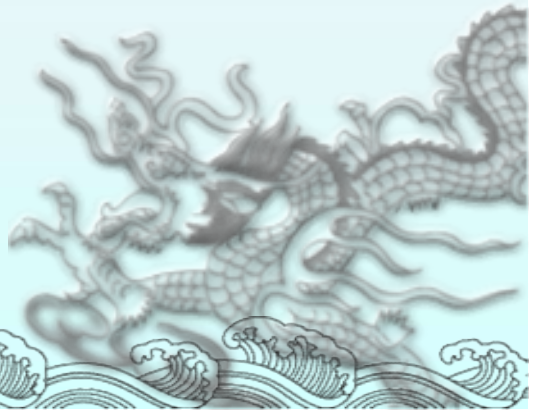
答案：**valid_month** 方法中对 **d.month** 的修改没有通过 **set** 方法，故为内容耦合。

内聚与耦合练习

例10：请判断下面例子属于哪种内聚或耦合

```
void validate_checkout_request(input_form i) {
        if(!valid_string(i.name)) {
                i.string = "Invalid name";
                error_message();
        }
        if(!valid_string(i.book)) {
                i.string = "Invalid book name";
                error_message();
        }
        valid_month(i.date);
}
void valid_month(Date d) {
        if(d.month < 1) {
                d.month = 1;
        }
        if(d.month > 12) {
                d.month = 12;
        }
        return 1;
}
```

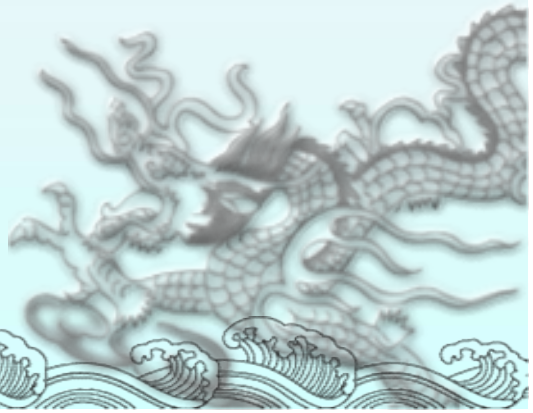
怎么改进?



内聚与耦合练习

例10：请判断下面例子属于哪种内聚或耦合

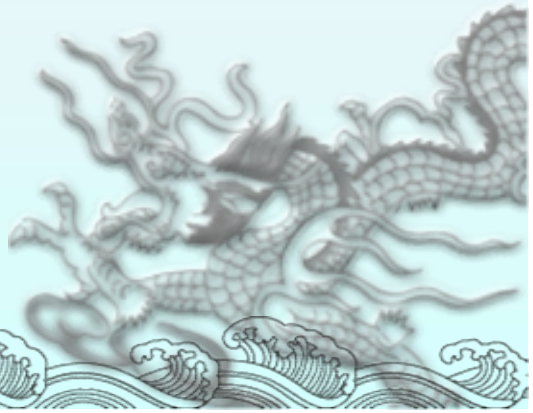
```
void validate_checkout_request(input_form i) {
        if(!valid_string(i.name)) {
                i.string = "Invalid name";
                error_message();
        }
        if(!valid_string(i.book)) {
                i.string = "Invalid book name";
                error_message();
        }
        valid_month(i.date);
}
void valid_month(Date d) {
        if(d.getMonth() < 1) {
                d.setMonth(1);
        }
        if(d.getMonth() > 12) {
                d.setMonth(12);
        }
        return;
}
```



内聚与耦合练习

例11：请判断下面例子属于哪种内聚或耦合

```
public class ScoreUtil {

        public static int getScore(boolean isGetAvg){
                int score = 0;
                if(isGetAvg){
                        //TODO get avg score
                }else{
                        //TODO get max score          }
                return score;
        }
}

public class TestA {

        private int getAvgScore(){
                return ScoreUtil.getScore(true);
        }
}
```



内聚与耦合练习

例11：请判断下面例子属于哪种内聚或耦合

```
public class ScoreUtil {

        public static int getScore(boolean isGetAvg){
                int score = 0;
                if(isGetAvg){
                        //TODO get avg score
                }else{
                        //TODO get max score          }
                return score;
        }
}

public class TestA {

        private int getAvgScore(){
                return ScoreUtil.getScore(true);
        }
}
```

答案：**TestA** 模块通过传入一个“做什么”的数据来控制 **ScoreUtil** 模块的流程，属于控制耦合。

内聚与耦合练习

例12：请判断下面例子属于哪种内聚或耦合

```
public class ScoreUtil {

        public static int getAvgScore(int score1, int score2){
                int sumScore = score1 + score2;
                return sumScore/2;
        }

}

public class TestA {

        private void printAvgScore(){
                int score1 = 50;
                int score2 = 100;
                System.out.println(ScoreUtil.getAvgScore(score1, score2));
        }
}
```

内聚与耦合练习

例12：请判断下面例子属于哪种内聚或耦合

```
public class ScoreUtil {

        public static int getAvgScore(int score1, int score2){
                int sumScore = score1 + score2;
                return sumScore/2;
        }

}

public class TestA {

        private void printAvgScore(){
                int score1 = 50;
                int score2 = 100;
                System.out.println(ScoreUtil.getAvgScore(score1,
score2));
        }
}
```

答案：模块**ScoreUtil**与模块**TestA**间存在数据耦合。

内聚与耦合练习

例13：请判断下面例子属于哪种内聚或耦合

```
public class Employee {
        // 雇员当前年龄
   public int curAge ;
   // 雇员还有多少年退休
   public int retireYear;
}

public class TestB {

   private void printEmployeeInfo(){
      Employee employeeA = new Employee();
      employeeA.curAge = 20;
      employeeA.retireYear = 65 - employeeA.curAge;

      System.out.println("employeeA curAge is ", employeeA.curAge);
      System.out.println("employeeA retireYear is ",
employeeA.retireYear);
   }
}
```

内聚与耦合练习

例13：请判断下面例子属于哪种内聚或耦合

```
public class Employee {
        // 雇员当前年龄
   public int curAge ;
   // 雇员还有多少年退休
   public int retireYear;
}

public class TestB {

   private void printEmployeeInfo(){
      Employee employeeA = new Employee();
      employeeA.curAge = 20;
      employeeA.retireYear = 65 - employeeA.curAge;

      System.out.println("employeeA curAge is ", employeeA.curAge);
      System.out.println("employeeA retireYear is ",
employeeA.retireYear);
   }
}
```

答案：类**Employee**与**TestB**间存在内容耦合。

内聚与耦合练习

例13：请判断下面例子属于哪种内聚或耦合

```
public class Employee {
        // 雇员当前年龄
   public int curAge ;
   // 雇员还有多少年退休
   public int retireYear;
}

public class TestB {

   private void printEmployeeInfo(){
      Employee employeeA = new Employee();
      employeeA.curAge = 20;
      employeeA.retireYear = 65 - employeeA.curAge;

      System.out.println("employeeA curAge is ", employeeA.curAge);
      System.out.println("employeeA retireYear is ",
employeeA.retireYear);
   }
}
```

怎么改进？

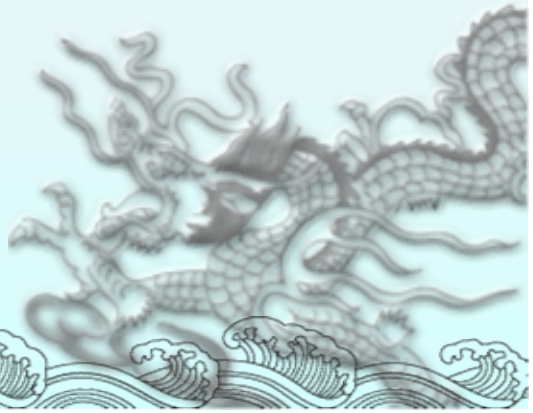
内聚与耦合练习

例13：请判断下面例子属于哪种内聚或耦合

```
public class Employee {
          // 雇员当前年龄
   private int curAge ;
   // 雇员还有多少年退休
   private int retireYear;
}

public class TestB {

   private void printEmployeeInfo(){
      Employee employeeA = new Employee();
      employeeA.setCurAge (20);
      employeeA.setRetireYear(65 - employeeA.getCurAge ());

      System.out.println("employeeA curAge is ", employeeA.curAge);
      System.out.println("employeeA retireYear is ",
employeeA.retireYear);
   }
}
```

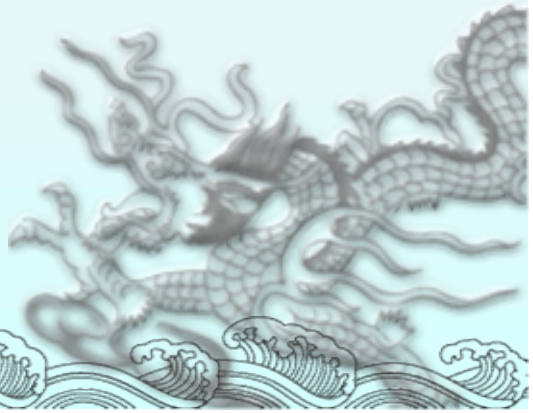
5.1.3抽象

- 所谓抽象就是将事物的相似方面集中和概括起来，暂时忽略它们之间的差异。或者说，抽象就是抽出事物的本质特性而暂时不考虑它们的细节。
- 在最高的抽象层次上，用自然语言，配合面向问题的专业术语，概括地描述问题的解法。
- 在中间的抽象层次上，采用过程化的描述方法。
- 在最底层，使用能够直接实现的方式来描述问题的解。



例如：开发一个CAD软件时的三种抽象层次

- 抽象层次Ⅰ. 用问题所处环境的术语来描述这个软件：
  该软件包括一个计算机绘图界面，向绘图员显示图形，以及一个数字化仪界面，用以代替绘图板和丁字尺。所有直线、折线、矩形、圆及曲线的描画、所有的几何计算、所有的剖面图和辅助视图都可以用这个CAD软件实现……。



- 抽象层次Ⅱ. 任务需求的描述。

CAD SOFTWARE TASKS

user interaction task;

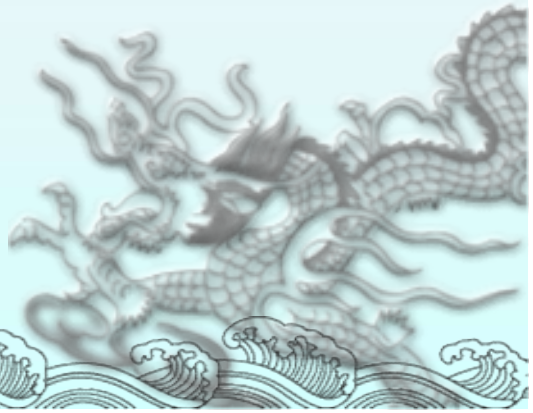
2-D drawing creation task;

graphics display task;

drawing file management task;

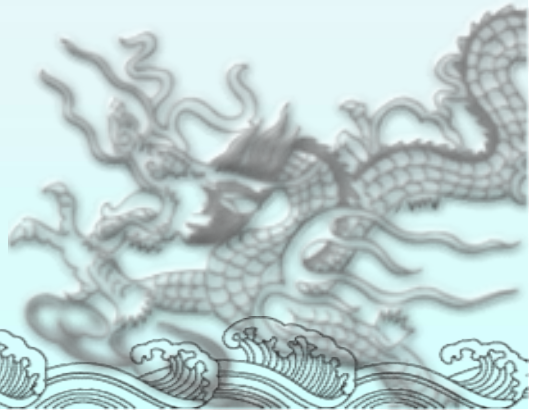
end.

在这个抽象层次上，未给出“怎样做”的信息，不能直接实现。



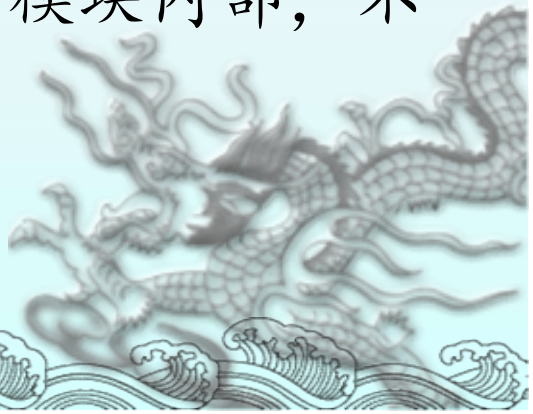
- 抽象层次Ⅲ. 程序过程表示。以2-D (二维)绘图生成任务为例:

```
PROCEDURE: 2-D drawing creation
REPEAT UNTIL (drawing creation task terminates)
   DO WHILE (digitizer interaction occurs)
      digitizer interface task;
        DETERMINE drawing request CASE;
            line:          line drawing task;
            rectangle: rectangle drawing task;
            circle:       circle drawing task;
            ……
```



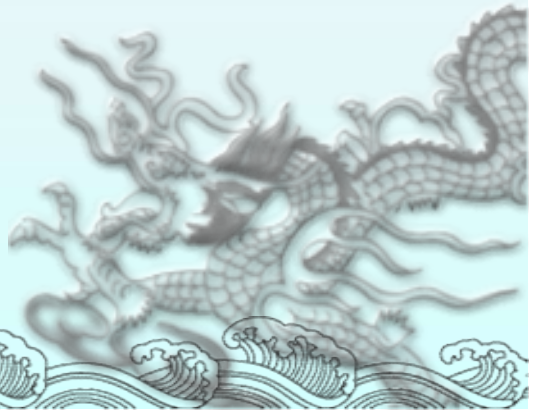
5.1.4信息隐藏

- 核心是：一个模块中所包含的信息，不允许其他不需要这些信息的模块访问。
- 模块化可以通过定义一组相互独立的模块来实现，这些独立的模块彼此间仅仅交换那些为完成相应功能而必须交换的信息。
- 信息隐藏对模块的过程细节和局部数据结构进行了屏蔽。
- 在设计模块时采取信息隐藏，使得大多数处理细节对软件的其他部分是隐蔽的。在将来修改软件时偶然引入错误所造成的影响就可以局限在一个或几个模块内部，不至于波及到软件其他部分。



5.1.5软件结构图

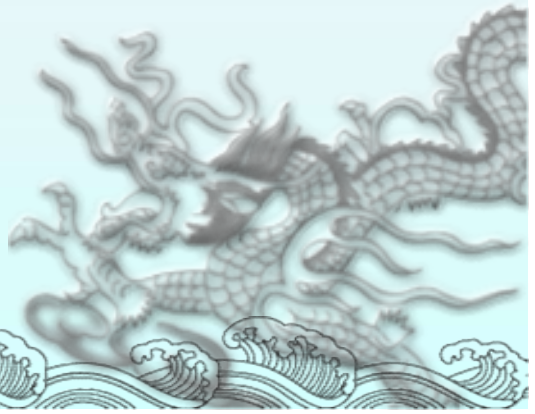
- 结构图反映程序中模块之间的层次调用关系和联系：它以特定的符号表示模块、模块间的调用关系和模块间信息的传递



① 模块：模块用矩形框表示，并用模块的名字标记它。

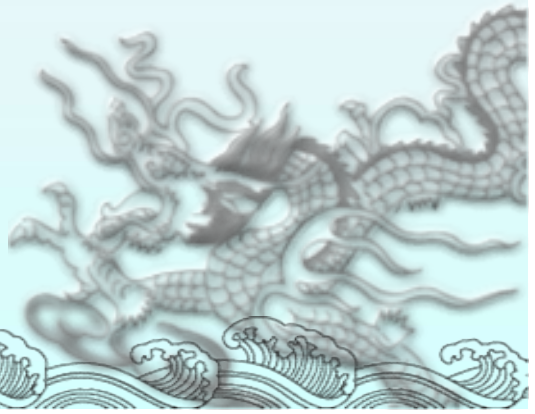
② 模块的调用关系

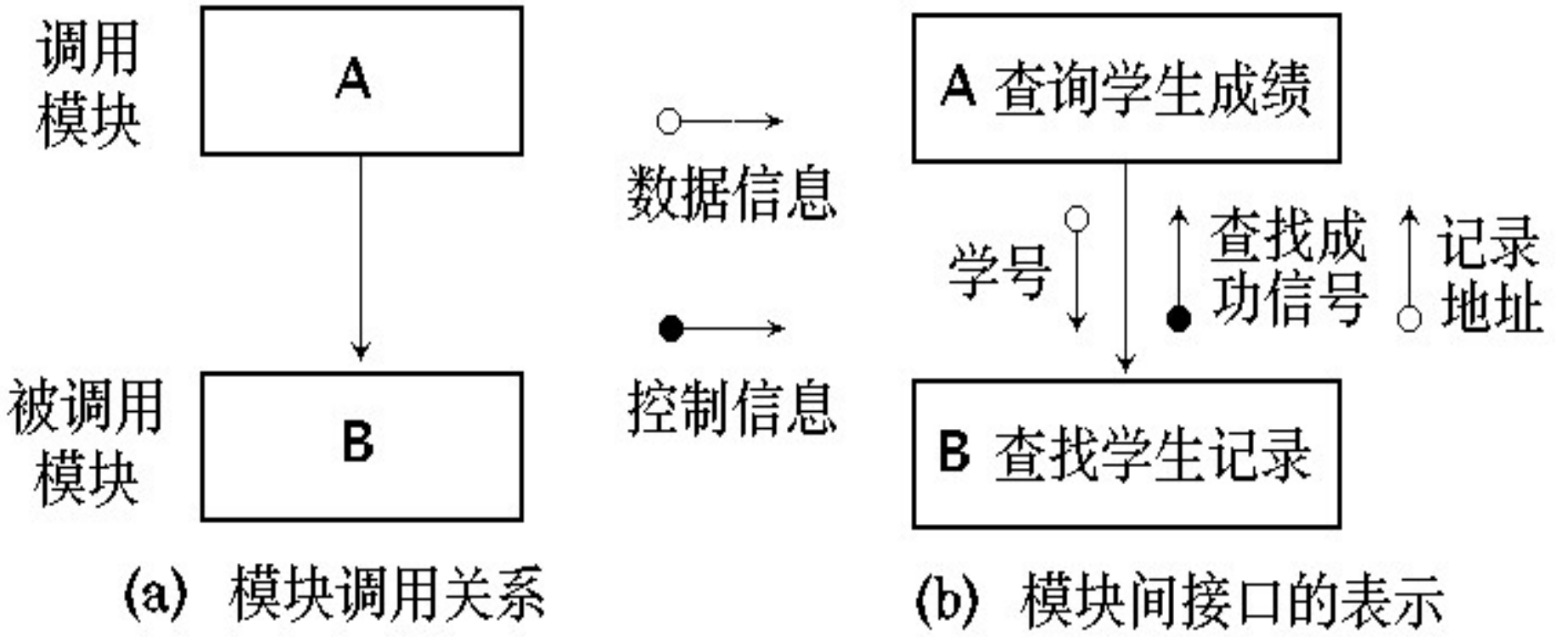
模块之间用单向箭头联结，箭头从调用模块指向被调用模块，表示调用模块调用了被调用模块。



③ 模块间的信息传递

当一个模块调用另一个模块时，调用模块把数据或控制信息传送给被调用模块，以使被调用模块能够运行。而被调用模块在执行过程中又把它产生的数据或控制信息回送给调用模块



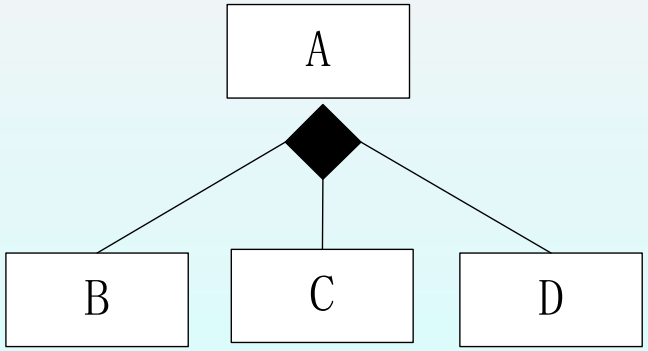


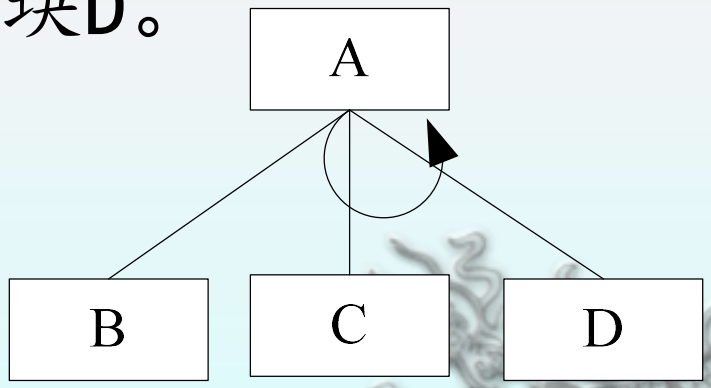
(a) 模块调用关系

(b) 模块间接口的表示

结构图说明（续）

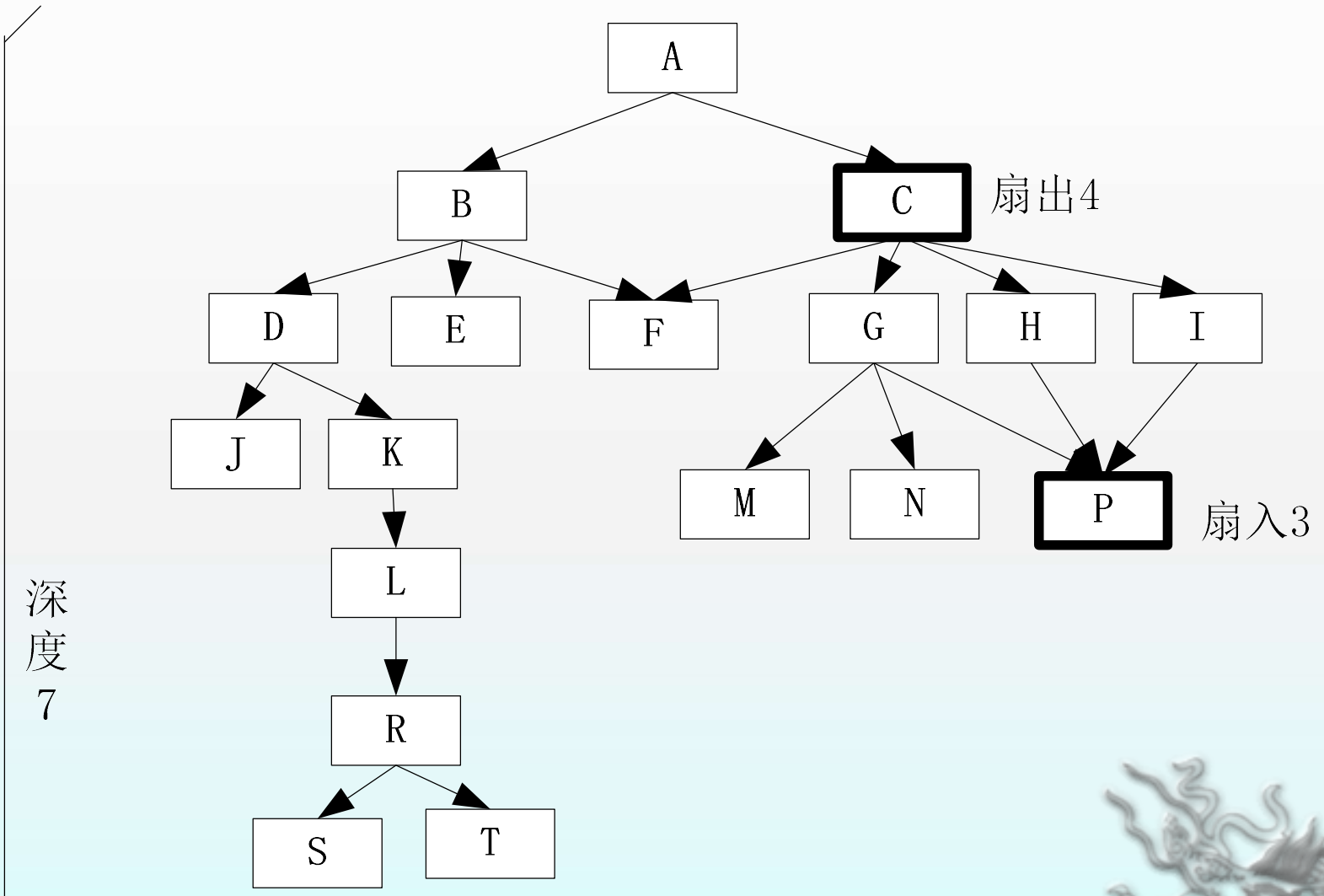
- 两个辅助符号：用符号◆表示一个模块有条件地调用另一个模块；用符号↻表示模块循环调用它的各下属模块。图中模块A下加一个菱形表示控制模块A按条件选择调用模块B、模块C、模块D。





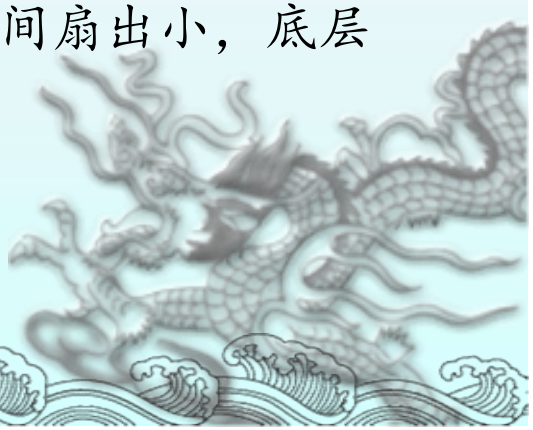
结构图说明（续）

- 结构图的形态特征：上层模块调用下层模块，模块自上而下“主宰”，自下而上“从属”。同一层的模块之间并没有这种主从关系。
- 结构图的深度：在多层次的结构图中，模块结构的层数称为该结构图的深度。下面结构图的深度为7。结构图的深度在一定意义上反映了程序结构的规模和复杂程度。对于中等规模的程序，结构图的深度约为10左右。对于一个大型程序，深度可以有几十层。
- 结构图的宽度：结构图中模块数最多的那层的模块个数称为结构图的宽度，下图的宽度为5。


A
B
C
扇出4
D
E
F
G
H
I
J
K
L
M
N
P
扇入3
R
S
T
深度7
宽度6

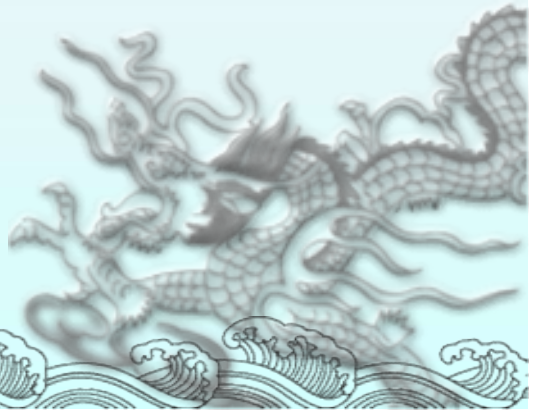
结构图说明（续）

- 模块的扇入和扇出：
  - 扇入：是指直接调用该模块的上级模块的个数。 扇入大表示模块的复用程度高。
  - 扇出：是指该模块直接调用的下级模块的个数。
  - 扇出大表示模块的复杂度高，需要控制和协调过多的下级模块；但扇出过小（例如总是1）也不好。
  - 扇出过大一般是因为缺乏中间层次，应该适当增加中间层次的模块。扇出太小时可以把下级模块进一步分解成若干个子功能模块，或者合并到它的上级模块中去。
  - 设计良好的软件结构，通常顶层扇出比较大，中间扇出小，底层模块则有大扇入。



内容提纲

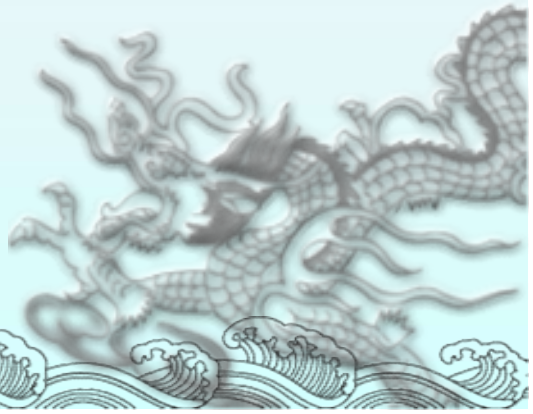
- 结构化软件设计
  - 软件设计的基本概念
  - 软件设计原则和影响设计的因素
  - 结构化设计方法
  - 优化软件结构设计



5.2软件设计原则和影响设计的因素

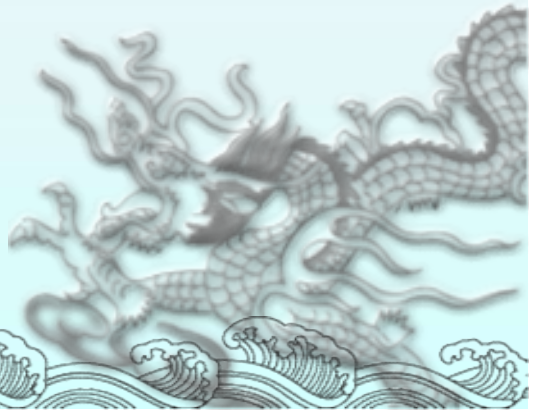
- 设计可回溯到需求。软件设计中的每个元素都可以对应到需求，保证设计使用户需要的。

- 充分利用已有的模块。一个复杂的软件通常是由一系列模块组成，很多模块可能在以前的系统中已经开发过了，如果这些模块设计得好，具有良好的可复用性，那么在设计新软件时应该尽可能使用已有的模块。



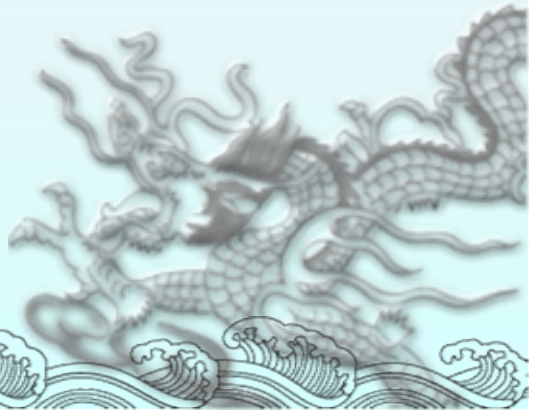
5.2软件设计原则和影响设计的因素

- 软件模块之间应该遵循高内聚、低耦合和信息隐藏的设计原则。
- 设计应该表现出一致性和规范性。在设计开始之前，设计小组应该定义设计风格和设计规范，保证不同的设计人员设计出风格一致的软件。



软件设计原则和影响设计的因素（续）

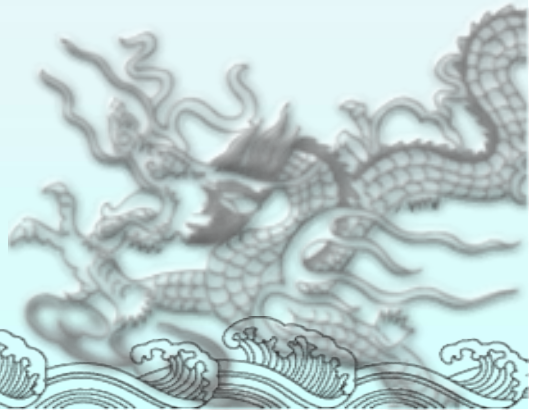
- 容错性设计。不管多么完善的软件都可能有潜在的问题，所以设计人员应该为软件进行容错性设计，当软件遇到异常数据、事件或操作时，软件不至于彻底崩溃。



软件设计原则和影响设计的因素（续）

- 设计的粒度要适当。设计不是编码，即使在详细设计阶段，设计模型的抽象级别也比源代码要高，它涉及的是模块内部的实现算法和数据结构。因此，不要用具体的程序代码取代设计。

- 在设计时就要开始评估软件的质量。软件的质量属性需要在设计时考虑如何实现，不要等全部设计结束之后再考虑软件的质量。

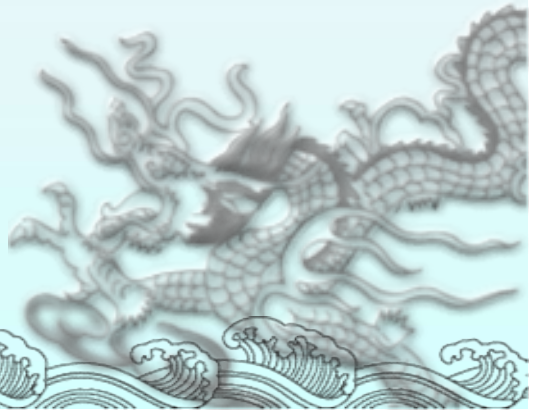


软件设计原则和影响设计的因素（续）

- 由多人共同设计一个软件时的协调问题；
- 设计人员的设计经验、理解力和喜好的差别；
- 一致的设计规范约束；
- 设计者的文化背景、信仰、价值观等其他方面的问题，这些都是影响软件设计的因素。

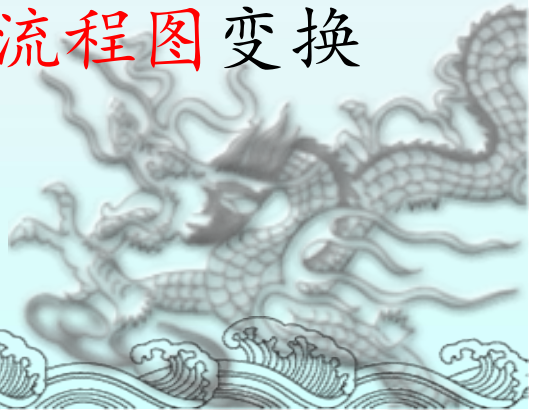
内容提纲

- 结构化软件设计
  - 软件设计的基本概念
  - 软件设计原则和影响设计的因素
  - 结构化设计方法
  - 优化软件结构设计



5.3 结构化设计方法

- 结构化设计方法通常也叫做面向数据流的设计或面向过程的设计。
- 结构化设计是基于模块化的、自顶向下、逐步求精等技术基础上的设计方法。
- 结构化设计与结构化分析和结构化编程方法前后呼应，形成了统一、完整的系列化方法。
- 结构化设计方法以需求分析阶段获得的数据流程图为基础，通过一系列映射，把数据流程图变换为软件结构图。



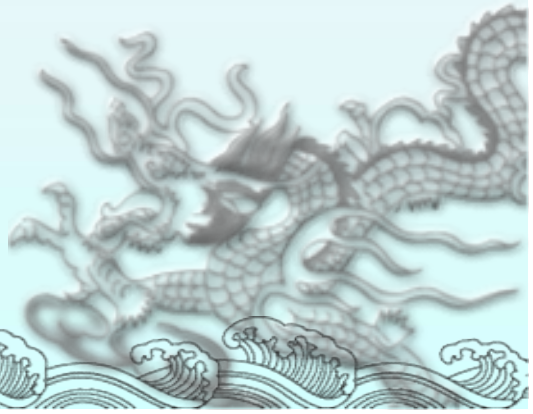
结构化设计方法4步骤

1）分析数据流的类型。数据流的类型有变换型和事务型两种，不同类型的数据流程图映射的软件结构有所不同。

2）将数据流程图映射为程序结构图。

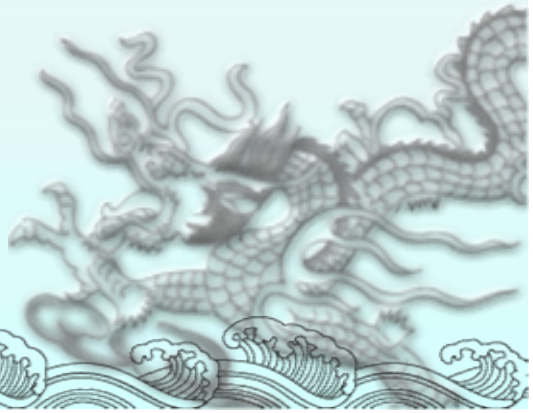
3）优化设计结构。

4）评审软件结构。



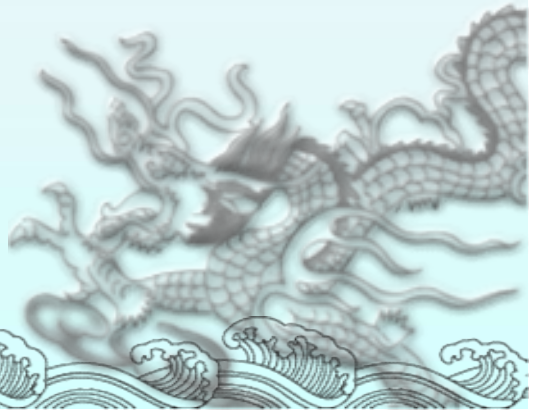
内容提纲

- 结构化软件设计
  - 软件设计的基本概念
  - 软件设计原则和影响设计的因素
  - 结构化设计方法
  - 优化软件结构设计



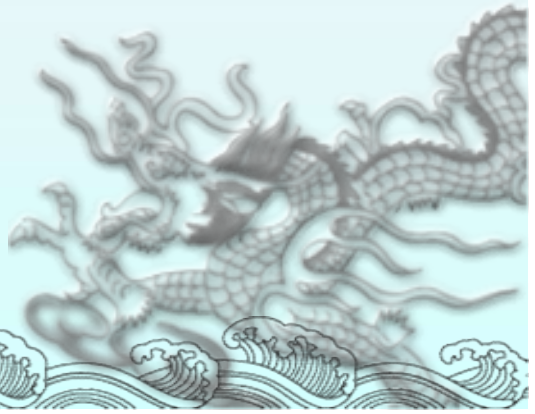
5.4 优化软件结构设计

- 数据流程图转换为软件结构图，应该对软件结构图进行优化。使其符合高内聚低耦合的、模块化、信息隐藏的原则。



5.4 优化软件结构设计（续）

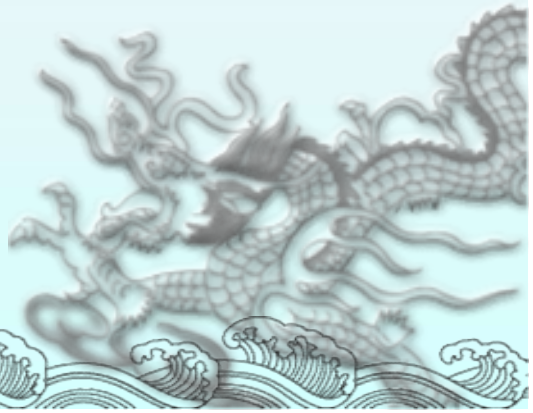
- 规则一：模块功能完善化。



5.4 优化软件结构设计（续）

- 规则二：设计功能单一和结果可预测的模块。

- 例如，如果在模块内部有一个局部控制变量M，在运行过程中模块的处理由这个控制变量确定，由于这个局部控制变量对于调用模块来说是隐蔽的，所以调用模块无法控制这个模块的执行，也不能预知将会引起什么后果，有可能造成混乱。

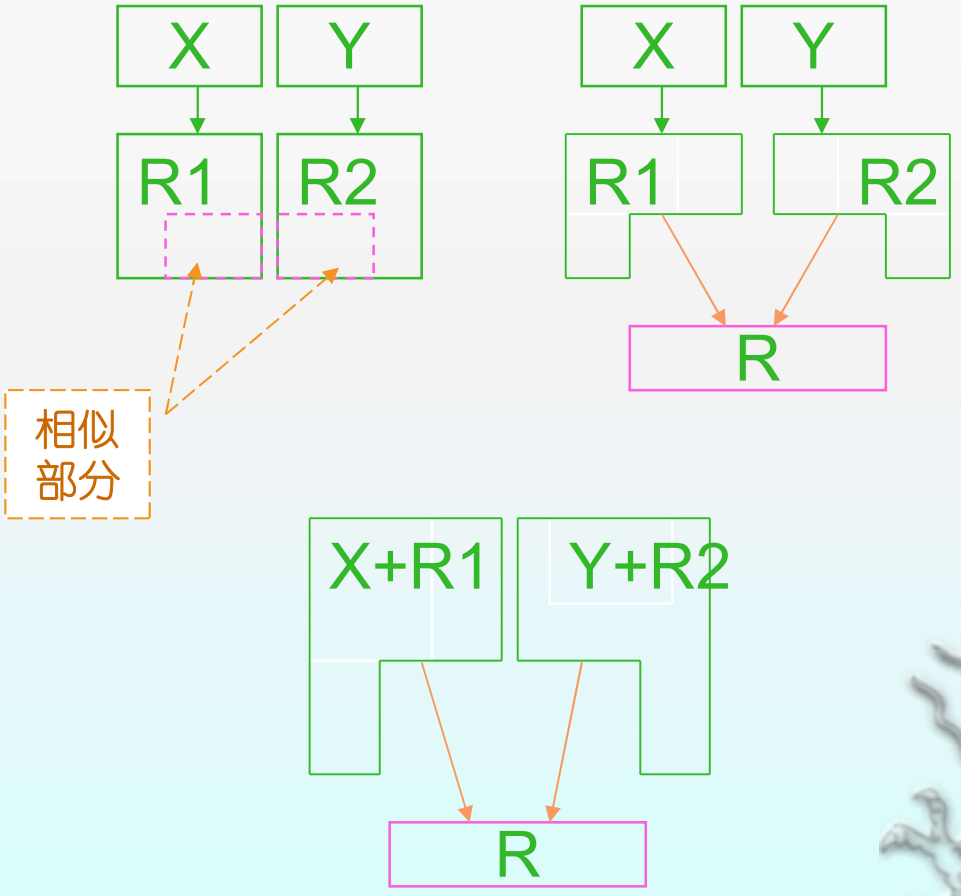


5.4 优化软件结构设计（续）

- 规则三：消除重复功能，改善软件结构。

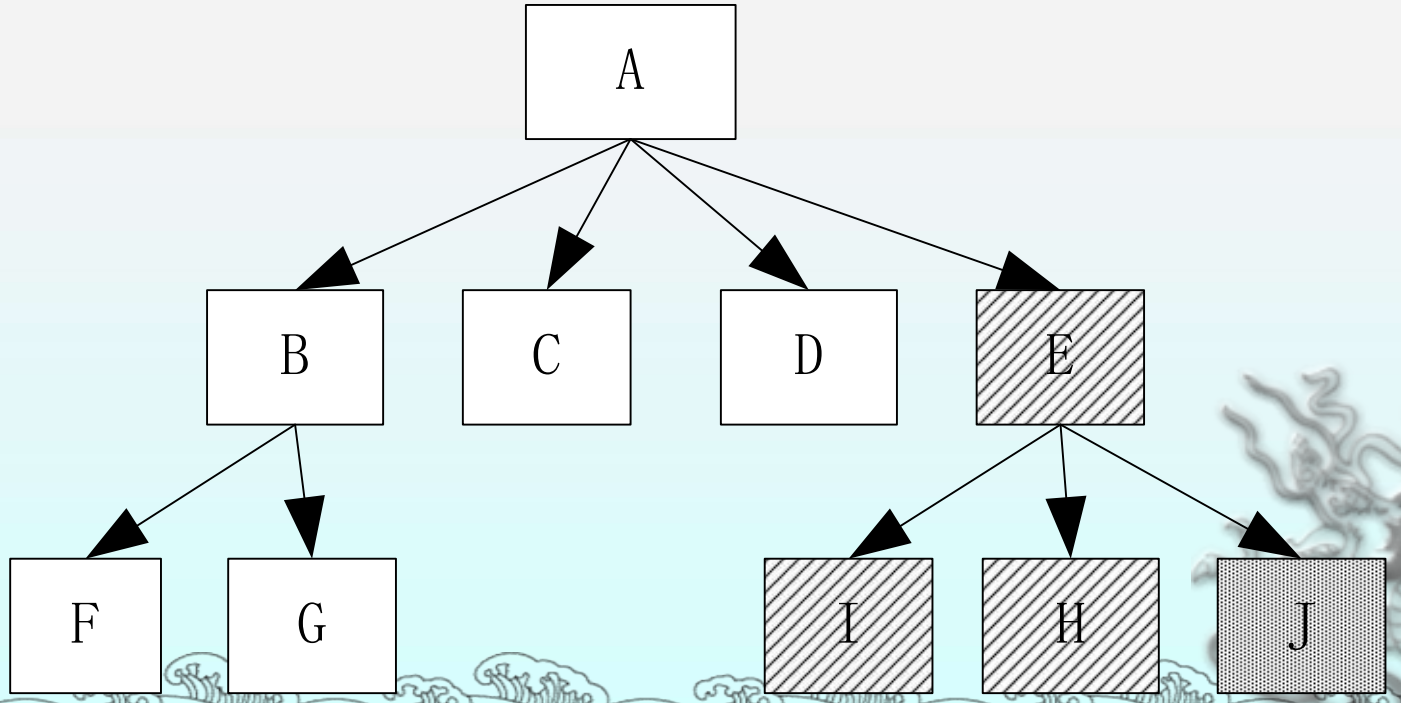
- 例如，当两个模块的功能完全相似，而只是所处理的数据类型不一致时，应该合并模块，同时修改模块的数据类型和变量定义。但是，如果两个模块的功能只是局部相似时，最好不要简单地合二为一，因为这种简单的并后会造成模块内部设置许多判断开关，模块的接口参数势必会传递一些控制信息，造成模块内聚降低。通常的处理办法是分析两个相似的模块，找出相同的部分，然后将相同的部分从分离出去组成一个新的模块。

相似模块合并方案示意图



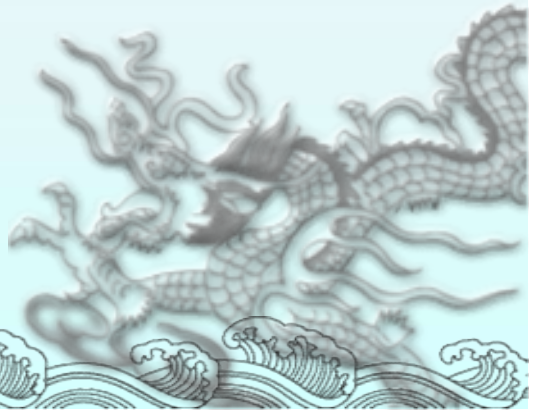
5.4 优化软件结构设计（续）

- 规则四：模块的作用范围应在控制范围之内。首先定义一个模块的控制范围是：是这个模块及其所有下属模块。
- 例如：图中模块E的控制范围是I、H、J。



5.4 优化软件结构设计（续）

- 一个模块的作用范围是：这个模块内判定的作用范围，凡是受这个判定影响的模块都属于这个判定的作用范围。
- 例如图中模块J的一个判定传递给E模块，然后再传递给I和H模块，这时模块J的作用范围是模块E、I、H、J。显然，这种设计是不好的，因为模块I和H不是模块J的控制范围，这样就导致模块之间传递的是控制参数，使模块之间的耦合增加。

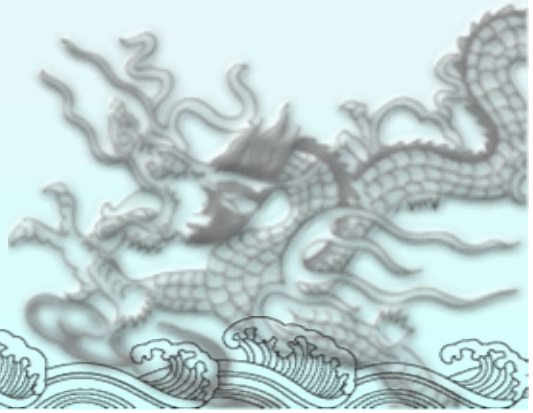


5.4 优化软件结构设计（续）

- 如果在设计过程中，发现作用范围不在控制范围内，可采用如下办法把作用范围移到控制范围之内：
- 1）提高控制模块的层次。将判定所在模块合并到父模块中，使判定处于较高层次。
- 2）将受判定影响的模块下移到控制范围内；
- 3）将判定上移到层次中较高的位置。但是要注意，判定所在的模块最好不要太高，模块之间的控制参数传递路径太长，增加了模块之间的耦合。比较好的方案是将判定提到模块E中。

5.4 优化软件结构设计（续）

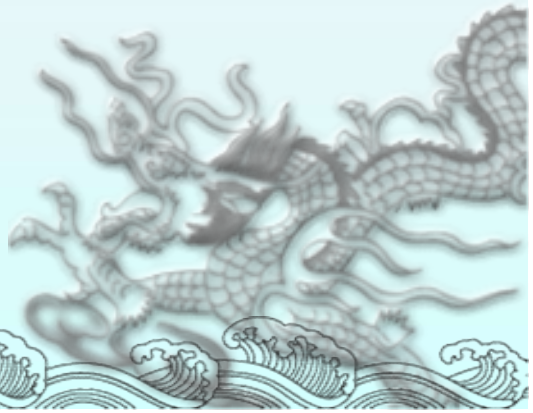
- 规则五：模块的大小要适中。模块的大小一般用模块的源代码数量来衡量，通常在设计过程中，将模块的源代码数量限制在５０～１００左右，即一页纸的范围内，这样阅读比较方便。



5.4 优化软件结构设计（续）

- 规则六：尽可能减少高扇出和高扇入的结构。

- 规则七：将模块中相对变化较大的部分剥离出去。为了加强模块的可复用性，在设计时将模块中相对稳定的部分与可能变化的部分分离，在分离的两个模块之间加一个接口模块对模块之间传递的参数进行整理，这对保持模块的稳定性和可重用性有很大作用。



5.4 优化软件结构设计（续）

- 对于有时间要求的软件结构，在整个设计阶段和编码阶段都必须进行优化，优化的方法如下：
  - 首先改进软件的结构。
  - 在详细设计时，挑出那些有可能占用过多时间的模块，为这些模块精心设计时间效率更高的处理算法。
  - 检测软件，分离出占用大量处理机资源的模块。如果有必要，用汇编语言或其它较低级的语言重新设计、编码，以提高软件的效率。

走查软件结构图

- 软件结构图中的模块关系体现的是调用关系，模块之间的接口参数在软件结构图上表现出来。
- 设计者根据调用关系在纸上对系统进行初步的试运行，方法是从软件结构图的最顶层按深度优先原则调用下级模块，直到图的最底层。在纸上试运行过程中，填写一张功能模块对照表。

数据结构与程序模块对照表

软件功能需求与程序模块对照表

功能	模块1	模块2	……	模块n
功能需求1	Module name			Module name
功能需求2	Module name	Module name		
……				
功能需求n	Module name			

第1列是分析阶段确定的软件功能编号，通常一个功能可能需要多个模块实现，如果模块超过5个，往往说明该功能太大，应该将其细分。

每个功能都应该有一条自上而下的模块调用通路，如果发现某条通路走下来不能实现需要的功能，就要重新检查数据流程图到软件结构图的转换是否正确。在走查模块时不要进入模块内部的具体处理算法，只是检查接口参数和分配的功能即可

用快速原型法修正设计

- 在设计时，有些问题很难确定是否能够实现，可以先开发一个原型，通过开发原型来发现设计中存在的问题，以便在编码之前解决很多棘手的问题。另外，原型可以促进开发人员之间、以及开发人员与用户之间的沟通。

- 开发原型时，通常忽略功能上的很多细节，只是将注意力放在系统的某个或某几个特定方面。

- 例如，界面方面、性能方面、还有安全方面等等，这种原型肯定会存在许多漏洞，但是，如果一个原型仅仅是要证明设计的可行性时，就不必太多的关注这些漏洞。这种原型属于抛弃型原型，意思是，开发的原型仅仅是为了证明系统某些特征的可行性，它不是最终的产品。

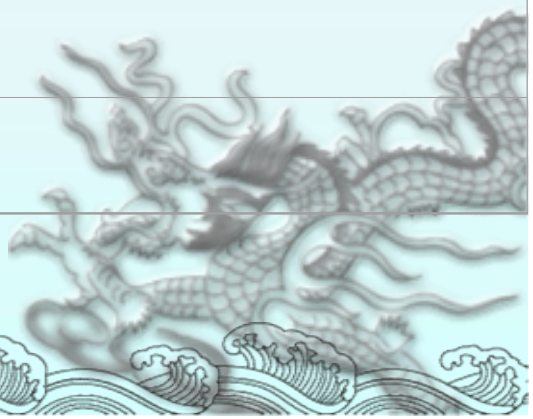
关于设计的说明

- 在程序结构被设计和优化后，应该对设计进行一些必要的说明。每个模块写一份处理说明；为模块之间的接口提供一份接口说明；确定全局数据结构；指出所有的设计约束和限制。
- 处理说明应该清楚地描述模块的主要处理任务、条件抉择和输人／输出。注意概要设计阶段不要对模块的内部处理过程进行详细描述，这项工作是详细设计的任务。
- 接口说明要给出一张表格，列出所有进入模块和从模块输出的数据。接口说明中应包括通过参数表传递的信息、对外界的输入／输出信息、访问全局数据区的信息等等。此外还要指出其下属的模块和上级模块。

模块说明表

模块名称：　　编号：

主要功能：	
输入参数及类型：	输出参数及类型：
上级调用模块：	
向下调用模块：	
局部数据结构：	
约束条件和设计限制：	



本章要点

- 软件设计的主要原则：模块独立性和信息隐藏。反映模块独立性的有两个标准：内聚和耦合。内聚衡量一个模块内部各个元素彼此结合的紧密程度，耦合衡量模块之间彼此依赖的程度。信息隐藏的核心内容是：一个模块中所包含的信息，不允许其他不需要这些信息的模块访问。
- 结构化设计是基于模块化的、自顶向下、逐步求精等概念上的设计方法。
- 结构化方法的流程：首先分析数据流的类型，将数据流程图映射为软件结构图，然后根据优化规则对于软件结构图进行设计优化。
- 数据设计包括：数据结构设计、文件设计和数据库设计。
- 详细设计是对模块内部的处理过程进行设计。