# 迪米特法则

## 定义

一个对象应该对其他对象保持最少的了解。

## 问题由来

**类与类之间的关系越密切，耦合度越大，当一个类发生改变时，对另一个类的影响也越大。**

## 解决方案

**尽量降低类与类之间的耦合。**

自从我们接触编程开始，就知道了软件编程的总的原则：**低耦合，高内聚**。

无论是面向过程编程还是面向对象编程，只有使各个模块之间的耦合尽量的低，才能提高代码的复用率。低耦合的优点不言而喻，但是怎么样编程才能做到低耦合呢？那正是迪米特法则要去完成的。

迪米特法则又叫最少知道原则，最早是在1987年由美国Northeastern University的Ian Holland提出。通俗的来讲，就是一个类对自己依赖的类知道的越少越好。也就是说，对于被依赖的类来说，无论逻辑多么复杂，都尽量地的将逻辑封装在类的内部，对外除了提供的public方法，不对外泄漏任何信息。

迪米特法则还有一个更简单的定义：只与直接的朋友通信。

首先来解释一下什么是直接的朋友：**每个对象都会与其他对象有耦合关系，只要两个对象之间有耦合关系，我们就说这两个对象之间是朋友关系。耦合的方式很多，依赖、关联、组合、聚合等。**

其中，我们称出现成员变量、方法参数、方法返回值中的类为**直接的朋友**，

而**出现在局部变量中的类则不是直接的朋友**。也就是说，**陌生的类最好不要作为局部变量的形式出现在类的内部。**

举一个例子：有一个集团公司，下属单位有分公司和直属部门，现在要求打印出所有下属单位的员工ID。

先来看一下违反迪米特法则的设计。

//总公司员工

class Employee{

private String id;

public void setId(String id){

this.id = id;

}

public String getId(){

return id;

}

}

//分公司员工

class SubEmployee{

private String id;

public void setId(String id){

this.id = id;

}

public String getId(){

return id;

}

}

class SubCompanyManager{

public List getAllEmployee(){

List list = new ArrayList();

for(int i=0; i<100; i++){

SubEmployee emp = new SubEmployee();

//为分公司人员按顺序分配一个ID

emp.setId("分公司"+i);

list.add(emp);

}

return list;

}

}

class CompanyManager{

public List getAllEmployee(){

List list = new ArrayList();

for(int i=0; i<30; i++){

Employee emp = new Employee();

//为总公司人员按顺序分配一个ID

emp.setId("总公司"+i);

list.add(emp);

}

return list;

}

public void printAllEmployee(SubCompanyManager sub){

List list1 = sub.getAllEmployee();

for(SubEmployee e:list1){

System.out.println(e.getId());

}

List list2 = this.getAllEmployee();

for(Employee e:list2){

System.out.println(e.getId());

}

}

}

public class Client{

public static void main(String[] args){

CompanyManager e = new CompanyManager();

e.printAllEmployee(new SubCompanyManager());

}

}

现在这个设计的主要问题出在CompanyManager中，根据迪米特法则，只与直接的朋友发生通信，而SubEmployee类并不是CompanyManager类的直接朋友（以局部变量出现的耦合不属于直接朋友），**从逻辑上讲总公司只与他的分公司耦合就行了，与分公司的员工并没有任何联系**，这样设计显然是增加了不必要的耦合。按照迪米特法则，应该避免类中出现这样非直接朋友关系的耦合。修改后的代码如下:

class SubCompanyManager{

public List getAllEmployee(){

List list = new ArrayList();

for(int i=0; i<100; i++){

SubEmployee emp = new SubEmployee();

//为分公司人员按顺序分配一个ID

emp.setId("分公司"+i);

list.add(emp);

}

return list;

}

**public void printEmployee(){**

List list = this.getAllEmployee();

for(SubEmployee e:list){

System.out.println(e.getId());

}

}

}

class CompanyManager{

public List getAllEmployee(){

List list = new ArrayList();

for(int i=0; i<30; i++){

Employee emp = new Employee();

//为总公司人员按顺序分配一个ID

emp.setId("总公司"+i);

list.add(emp);

}

return list;

}

public void printAllEmployee(SubCompanyManager sub){

**sub.printEmployee();**

List list2 = this.getAllEmployee();

for(Employee e:list2){

System.out.println(e.getId());

}

}

}

修改后，为分公司增加了打印人员ID的方法，总公司直接调用来打印，**从而避免了与分公司的员工发生耦合。**

迪米特法则的初衷是降低类之间的耦合，由于每个类都减少了不必要的依赖，因此的确可以降低耦合关系。但是凡事都有度，虽然可以避免与非直接的类通信，但是要通信，必然会通过一个"中介"来发生联系，例如本例中，总公司就是**通过分公司这个"中介"来与分公司的员工发生联系的。**

过分的使用迪米特原则，会产生大量这样的中介和传递类，导致系统复杂度变大。所以在采用迪米特法则时要反复权衡，既做到结构清晰，又要高内聚低耦合。

**其实在我看来，就是各自的功能在各自所属的类中实现（高内聚），其他的类只要调用就可以（低耦合）。**

像上面的例子，分公司的员工属于分公司的类，就尽量不要出现在总公司中，总公司只要调用分公司的方法即可。