迪米特法则 慈禧太后为何不和陌生人说话

应用场景举例：

在《投名状》这部轰动一时的影片中有这么一个片段，慈禧太后召见庞青龙，带路的太监说，从门口到见到老佛爷（也就是慈禧太后）这条短短的路他花了大半辈子才走完，而很多人一辈子也走不完，感叹道：“你倒好，这么短的时间里就走了别人花费一生才能走完的道路”。

定义：

迪米特法则(Law of Demeter，简写 LoD)又叫做最少知识原则（Least Knowledge Principle 简写 LKP），也就是说，一个对象应当对其他对象尽可能少的了解，不和陌生人说话。

迪米特法则最初是用来作为面向对象的系统设计风格的一种法则，于 1987 年秋天由 lan holland 在美国东北大学为一个叫做迪米特的项目设计提出的。被 UML 的创始者之一 Booch 等普及。后来，因为在经典著作《 The Pragmatic Programmer》阐述而广为人知。

狭义的迪米特法则是指：如果两个类不必彼此直接通信，那么这两个类就不应当发生直接的相互作用。如果其中一个类需要调用另一类的某一个方法的话，可以通过第三者转发这个调用。

广义的迪米特法则是指：一个模块设计的好坏的一个重要标志就是该模块在多大程度上讲自己的内部数据与实现的有关细节隐藏起来。

一个软件实体应当尽可能少的与其他实体发生相互作用。

每一个软件单位对其他的单位都只有最少的知识，而且局限于那些与本单位密切相关的软件单位。

迪米特法则的目的在于降低类与类之间的耦合。由于每个类尽量减少对其他类的依赖，因此，很容易使得系统的功能模块功能独立，是的相互间存在尽可能少的依赖关系。

在运用迪米特法则到系统的设计中时,要注意以下几点:

第一：在类的划分上，应当创建弱耦合的类，类与类之间的耦合越弱，就越有利于实现可复用的目标。

第二：在类的结构设计上，每个类都应该降低成员的访问权限。

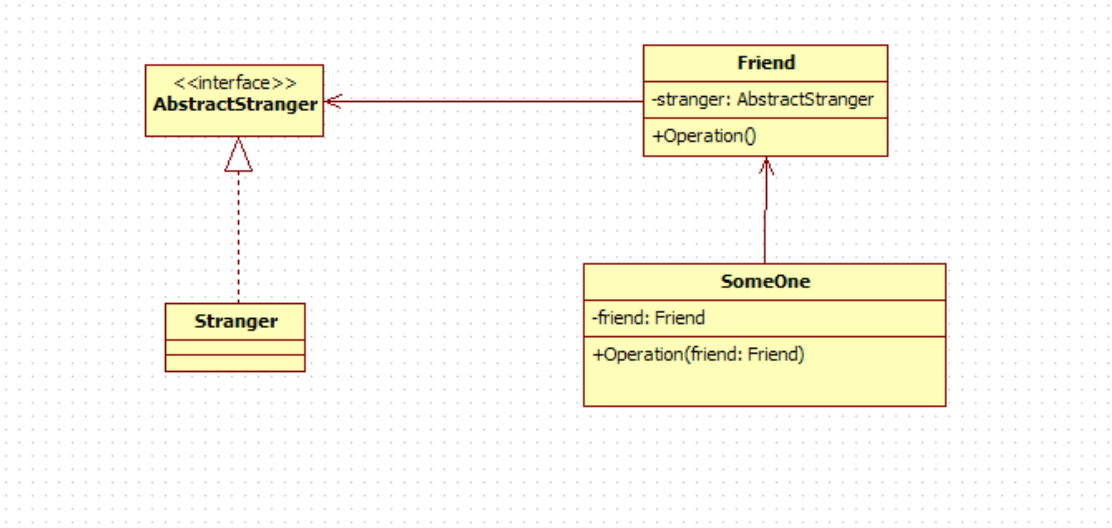
第三：在类的设计上，只要有可能，一个类应当设计成不变的类。

第四：在对其他类的应用上，一个对象对其他类的对象的应用应该降到最低。

第五：尽量限制局部变量的有效范围。

但是过度使用迪米特法则，也会造成系统的不同模块之间的通信效率降低，使系统的不同模块之间不容易协调等缺点。同时，因为迪米特法则要求类与类之间尽量不直接通信，如果类之间需要通信就通过第三方转发的方式，这就直接导致了系统中存在大量的中介类，这些类存在的唯一原因是为了传递类与类之间的相互调用关系，这就毫无疑问的增加了系统的复杂度。解决这个问题的方式是：使用依赖倒转原则（通俗的讲就是要针对接口编程，不要针对具体编程），这要就可以是调用方和被调用方之间有了一个抽象层，被调用方在遵循抽象层的前提下就可以自由的变化，此时抽象层成了调用方的朋友。

如下图所示：



故事分析：

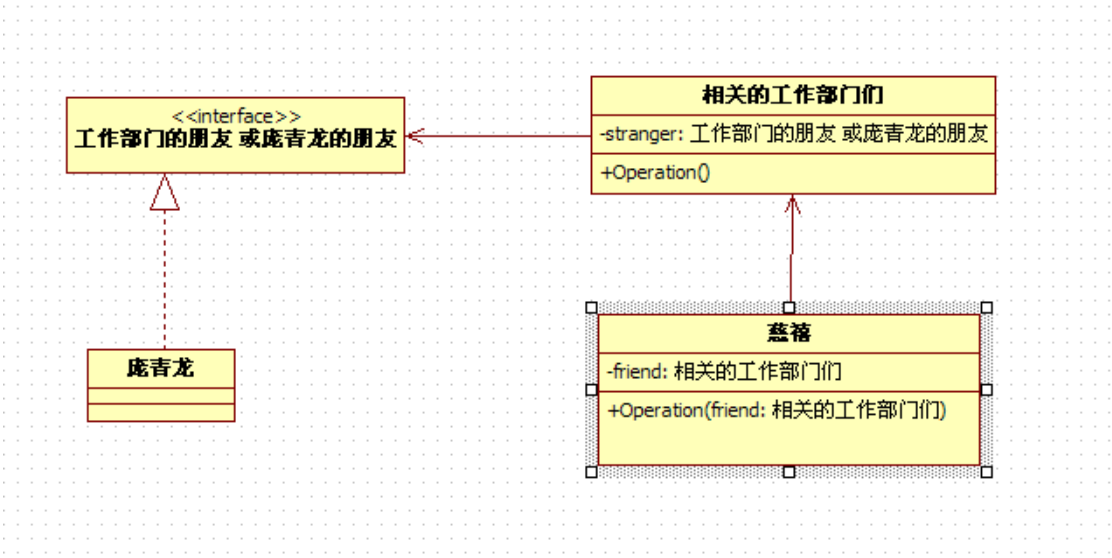
慈禧太后要召见庞青龙。庞青龙在见到慈禧太后前经历了那些过程呢？首先，当然是有人通知庞青龙要被召见，通知庞青龙的人当然不会是慈禧本人！慈禧只是下达旨意，然后又相关的只能部门传达旨意，相关部门的领导人也不会亲自去通知庞青龙，这些领导人会派遣信得过的人去，而这个被派遣的人也不是说想见庞青龙就能见得了的，他也必须通过和庞青龙熟悉的人，最后才能见到庞青龙，从而才能成功的传达旨意；第二：在进宫前，庞青龙必须卸掉自己随身携带的任何武器；第三：会有专门的只能部门对庞青龙进行全身彻底的检查，以防有任何可以伤害人的东西携带在身上，当然这个过程可能非常的复杂和繁琐。最后，由一个太监带路到慈禧面前。当然，见到慈禧的时候，庞青龙不是和慈禧坐在一起的，要报仇距离！慈禧也深深的懂得保持距离的重要性！

见到慈禧太后以后慈禧也没有和庞青龙直接说话，因为慈禧不和陌生人说话！而是同时身边的人传达自己的话，慈禧只需颐指气使即可。

从上面的过程中我们可以看出处处体现了迪米特法则的应用，慈禧知道庞青龙这个人肯定是通过一层又一层的关系得知的，就是迪米特法则中的第三者转发而且这里面说不定还有若干个第三者的转发！而从慈溪下旨召见庞青龙到庞青龙收到旨意，这中间又是完美的提现了迪米特法则，这中间经历无数的第三者！就连庞青龙面见到慈禧后，慈禧也不和他直接说话，而是通过身边的人传话，这慈禧是不是太傻了，直接和他说不就行了吗?慈禧当然不傻，因为她深知迪米特法则的重要。两个类的对象之间如果不发生直接的联系就不直接发生关系！

不过这也产生了一个问题，这中间经历这么多的转发，需要机构和人啊？或许这就是为什么当时的清政府机构那么庞大、财政开支惊人的原因之一吧^_^

如下图所示：



Java 代码实现：

新建一个陌生人的抽象父类，其他的陌生人继承这个接口：

```
package com.diermeng.designPattern.LoD;
/*
 * 抽象的陌生人类
 */
public abstract class Stranger {
    /*
     * 抽象的行为方法
     */
    public abstract void operation();
}
```

庞青龙实现继承实现陌生人抽象类：

```
package com.diermeng.designPattern.LoD.impl;
```

```
import com.diermeng.designPattern.LoD.Stranger;
/*
 * 庞青龙对抽象类Stranger的实现
 */
public class PangQingyong extends Stranger{
    /*
     * 操作方法
     * @see com.diermeng.designPattern.LoD.Stranger#operation()
     */
    public void operation(){
        System.out.println("禀报太后：我是庞青龙，我擅长用兵打仗！");
    }
}
```

朋友类，这里指太监类

```
package com.diermeng.designPattern.LoD.impl;

import com.diermeng.designPattern.LoD.Stranger;
/*
 * 太监类
 */
public class Taijian {
    /*
     * 太监类的操作方法
     */
    public void operation(){
        System.out.println("friends paly");
    }

    /*
     * 由太监类提供Cixi需要的方法
     */
    public void findStranger() {
        //创建一个Stranger
        Stranger stranger = new PangQingyong();
        //执行相应的方法
        stranger.operation();
    }
}
```

调用者类的代码，在这里是慈禧太后类：

```
package com.diermeng.designPattern.LoD.impl;

/*
 * 慈禧类
 */
public class Cixi {
    //拥有对太监的引用，即对"朋友"的引用
    private Taijian taijian;

    //得到一个太监对象
    public Taijian getTaijian() {
        return taijian;
    }

    //设置一个太监对象
    public void setTaijian(Taijian taijian) {
        this.taijian = taijian;
    }

    //操作方法
    public void operation(){
        System.out.println("someone play");
    }


}
```

建立一个测试类，代码如下：

```
package com.diermeng.designPattern.LoD.client;

import com.diermeng.designPattern.LoD.impl.Taijian;
import com.diermeng.designPattern.LoD.impl.Cixi;

/*
 * 测试客户端
 */
public class LoDTest {

    public static void main(String[] args) {
        //声明并实例化慈禧类
```

```
        Cixi zhangsan = new Cixi();

        //设置一个太监实例化对象，即找到一个"朋友"帮忙做事
        zhangsan.setTaijian(new Taijian());

        //慈禧通过宫中太监传话给陌生人
        zhangsan.getTaijian().findStranger();
    }
}
```

程序运行结果如下：

```
禀报太后：我是庞青龙，我擅长用兵打仗！
```

已有应用简介：

迪米特法则或者最少知识原则作为面向对象设计风格的一种法则，也是很多著名软件设计系统的指导原则，比如火星登陆软件系统、木星的欧罗巴卫星轨道飞船软件系统。

温馨提示：

迪米特法则是一种面向对象系统设计风格的一种法则，尤其适合做大型复杂系统设计指导原则。但是也会造成系统的不同模块之间的通信效率降低，使系统的不同模块之间不容易协调等缺点。同时，因为迪米特法则要求类与类之间尽量不直接通信，如果类之间需要通信就通过第三方转发的方式，这就直接导致了系统中存在大量的中介类，这些类存在的唯一原因是为了传递类与类之间的相互调用关系，这就毫无疑问的增加了系统的复杂度。解决这个问题的方式是：使用依赖倒转原则（通俗的讲就是要针对接口编程，不要针对具体编程），这要就可以是调用方和被调用方之间有了一个抽象层，被调用方在遵循抽象层的前提下就可以自由的变化，此时抽象层成了调用方的朋友。