1. 阅读：JavaSE Application Design With MVC

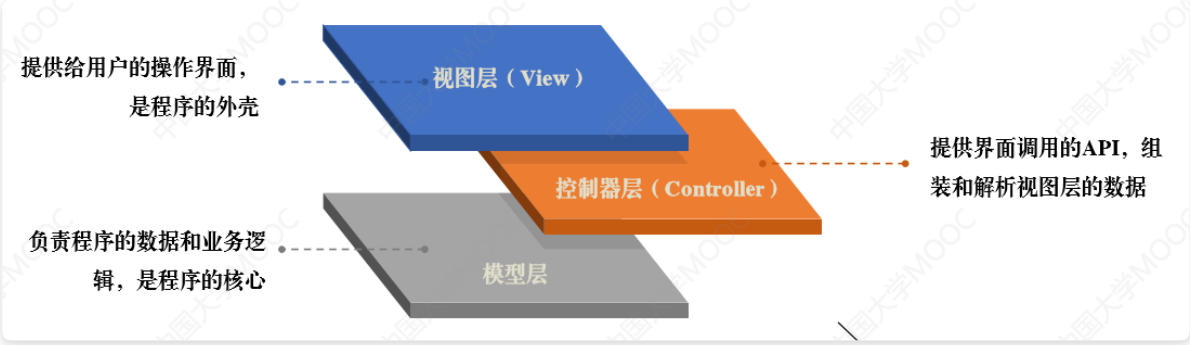
简单来说，软件模块化让一个复杂的软件系统可以通过多人协作或一步一步的实现来完成整个系统的开发

1.1 从两层划分到三层MVC

软件最简单的一种划分软件的方法是分为两层：

- **视图层View**：包含了界面长什么样子、数据如何输入和输出，是系统的外壳
- **模型层Model**：负责整个系统的逻辑、数据的存储，是整个系统的核心

其中视图层比较直接，一般使用按照功能的方式去设计；模型层比较复杂，一般使用面向对象的方式来设计。



两层划分

但大多数情况下，视图层和模型层之间差异较大，可以增加一个中间层——**控制器层Controller**，如此，便可形成我们所熟悉的MVC三层架构

控制器层可以把面向对象的设计转换成为面向功能所需要的的API，同时控制器层也负责把数据从视图层转换为逻辑层所需要的对象

1.2 更加符合实际情况的层次划分

在实际运用中，模型层往往相当复杂，根据逻辑不通，可以把复杂的模型层细分为若干个更小的层次，还可划分出以下层：

- **服务层Service**：负责处理业务逻辑
- **数据访问层Dao**：负责获取数据，并组装成对象
- **映射层Mapper**：负责将数据对象模型映射为数据库关系模型
- **实体层Entity**：与模型有关的代码

更复杂的分层方式

2. LoD原则强调“只和朋友通信，不和陌生人说话”。请举例说明 “朋友圈” 认定依据是啥?

2.1 LoD原则简单说明

迪米特法则(Law of Demeter, LoD)：一个对象应该对其他对象保持最少的了解，又叫最少知道原则(Least Knowledge Principle, LKP)。

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。如果其中的一个类需要调用另外一个类的某一个方法的话，可以通过第三者转发这个调用，尽量降低类与类之间的耦合。

这是高内聚低耦合的体现，强调只和朋友交流，不和陌生人说话。保持神秘。

2.2 朋友圈的确定

如果满足以下条件，可将其划入朋友圈：

1. 当前对象本身（this）
2. 以参数方式传入到当前对象方法中的对象
3. 当前对象的实例变量直接引用的变量
4. 当前对象的实例变量如果是一个聚集，那么聚集中的元素也都是朋友
5. 当前对象所创建的对象

2.3 LoD简单举例

2.3.1 情景说明

我们可以假设这样一个场景：老师**(Teacher类)**需要给**学生(Student类)**布置一些作业**(Homework类)**，这样一共有三个类：老师、作业、学生。其中，老师和学生关系紧密、学生和作业关系紧密，那么如果老师需要检查作业，应该调用学生类的某个方法或与学生类发生关联，而不应该与作业类发生关联。

2.3.2 错误例说明

作业类Homework

```
package LoD.InstanceWrong;

public class Homework {
    String subjectName;

    public Homework(String subjectName) {
        this.subjectName = subjectName;
    }

    public String getSubjectName() {
        return subjectName;
    }
}
```

学生类Student

```
package LoD.InstanceWrong;

import java.util.List;

public class Student {
    String studentName;

    public Student(String studentName) {
        this.studentName = studentName;
    }

    public String getStudentName() {
        return studentName;
    }

    public void receiveHomework(List<Homework> homeworkList) {
        System.out.printf("%s说：一共布置了%d门作业，它们是：\n", this.studentName,
 homeworkList.size());
        for( Homework homework : homeworkList) {
            System.out.println(homework.getSubjectName());
        }
    }
}
```

老师类Teacher

```
package LoD.InstanceWrong;

import java.util.ArrayList;
import java.util.List;

public class Teacher {
    String teacherName;

    public Teacher(String teacherName) {
        this.teacherName = teacherName;
    }

    // 老师给某个学生布置作业
    public void arrangeHomework(Student student) {
```

```
        Homework chinese = new Homework("语文");
        Homework math = new Homework("数学");
        Homework english = new Homework("英语");
        List<Homework> homeworkList = new ArrayList<Homework>();
        homeworkList.add(chinese);
        homeworkList.add(math);
        homeworkList.add(english);
        System.out.printf("%s老师已给%s布置作业\n", this.teacherName,
student.getStudentName());
        student.receiveHomework(homeworkList);
    }
}
```

测试类TestMethod

```
package LoD.InstanceWrong;

public class TestMethod {
    public static void main(String[] args) {
        Teacher teacher = new Teacher("李贵林");
        Student student = new Student("小明");
        teacher.arrangeHomework(student);
    }
}
```

运行结果：

> "C:\Program Files\Java\jdk-11.0.11\bin\java.exe" "-javaagent:E:\IntelliJ IDEA 2021.1.2\lib\idea_rt.jar=5794:E:\IntelliJ IDEA 2021.1.2\bin" -Dfile.encoding=UTF-8 -classpath E:\java\Architecture\out\production\Architecture LoD.InstanceWrong.TestMethod
> 李贵林老师已给小明布置作业
> 小明说：一共布置了3门作业，它们是：
> 语文
> 数学
> 英语
>
> Process finished with exit code 0

可以看到，这个例子中，Teacher与Homework和Student都发生了关联，耦合度比较高。如果之后需要对相关功能进行修改，会比较麻烦。

2.3.3 正确例说明

作业类Homework

```java
package LoD.Instance;

public class Homework {
    String subjectName;

    public Homework(String subjectName) {
        this.subjectName = subjectName;
    }

    public String getSubjectName() {
        return subjectName;
    }
}
```

学生类Student

```java
package LoD.Instance;

import java.util.ArrayList;
import java.util.List;

public class Student {
    String studentName;

    public Student(String studentName) {
        this.studentName = studentName;
    }

    public String getStudentName() {
        return studentName;
    }

    public void receiveHomework(List<String> homeworkList) {
        List<Homework> homeworkList1 = new ArrayList<Homework>();
        System.out.printf("%s说：一共布置了%d门作业，它们是：\n", this.studentName,
 homeworkList.size());
        for (String str : homeworkList) {
            System.out.println(str);
            Homework homework = new Homework(str);
            homeworkList1.add(homework);
        }

    }
}
```

老师类Teacher

```java
package LoD.Instance;

import java.util.ArrayList;
import java.util.List;

public class Teacher {
    String teacherName;

    public Teacher(String teacherName) {
```

```
        this.teacherName = teacherName;
    }

    // 老师给某个学生布置作业
    public void arrangeHomework(Student student) {
        List<String> homeworkList = new ArrayList<String>(3);
        homeworkList.add("语文");
        homeworkList.add("数学");
        homeworkList.add("英语");
        System.out.printf("%s老师已给%s布置作业\n", this.teacherName,
student.getStudentName());
        student.receiveHomework(homeworkList);
    }
}
```

测试类TestMethod

```
package LoD.Instance;

public class TestMethod {
    public static void main(String[] args) {
        Teacher teacher = new Teacher("李贵林");
        Student student = new Student("小明");
        teacher.arrangeHomework(student);
    }
}
```

运行结果：

> "C:\Program Files\Java\jdk-11.0.11\bin\java.exe" "-javaagent:E:\IntelliJ IDEA 2021.1.2\lib\idea_rt.jar=5898:E:\IntelliJ IDEA 2021.1.2\bin" -Dfile.encoding=UTF-8 -classpath E:\java\Architecture\out\production\Architecture LoD.Instance.TestMethod
> 李贵林老师已给小明布置作业
> 小明说：一共布置了3门作业，它们是：
> 语文
> 数学
> 英语
>
> Process finished with exit code 0

这样，Teacher不再与Homework发生关联，降低了类之间的耦合度