###### Java EE应用的分层模型

* Domain Object（领域对象）层。

此层是由一系列的POJO（Plain Old Java Object，普通的、传统的Java对象）组成，这些对象是该系统的Domain Object，往往包含了各自所需实现的业务逻辑方法。

* **DAO（Data Access Object，数据访问对象）层。**

此层由一系列的DAO组件组成，这些DAO实现了对数据库的创建、查询、更新和删除（CRUD）等原子操作。在经典JavaEE应用中，DAO层也被改称为EAO层，EAO层组件的作用与DAO层组件的作用基本相似。只是EAO层主要完成对实体（Entity）的CRUD操作，因此简称为EAO层。

* Service（业务逻辑）层。

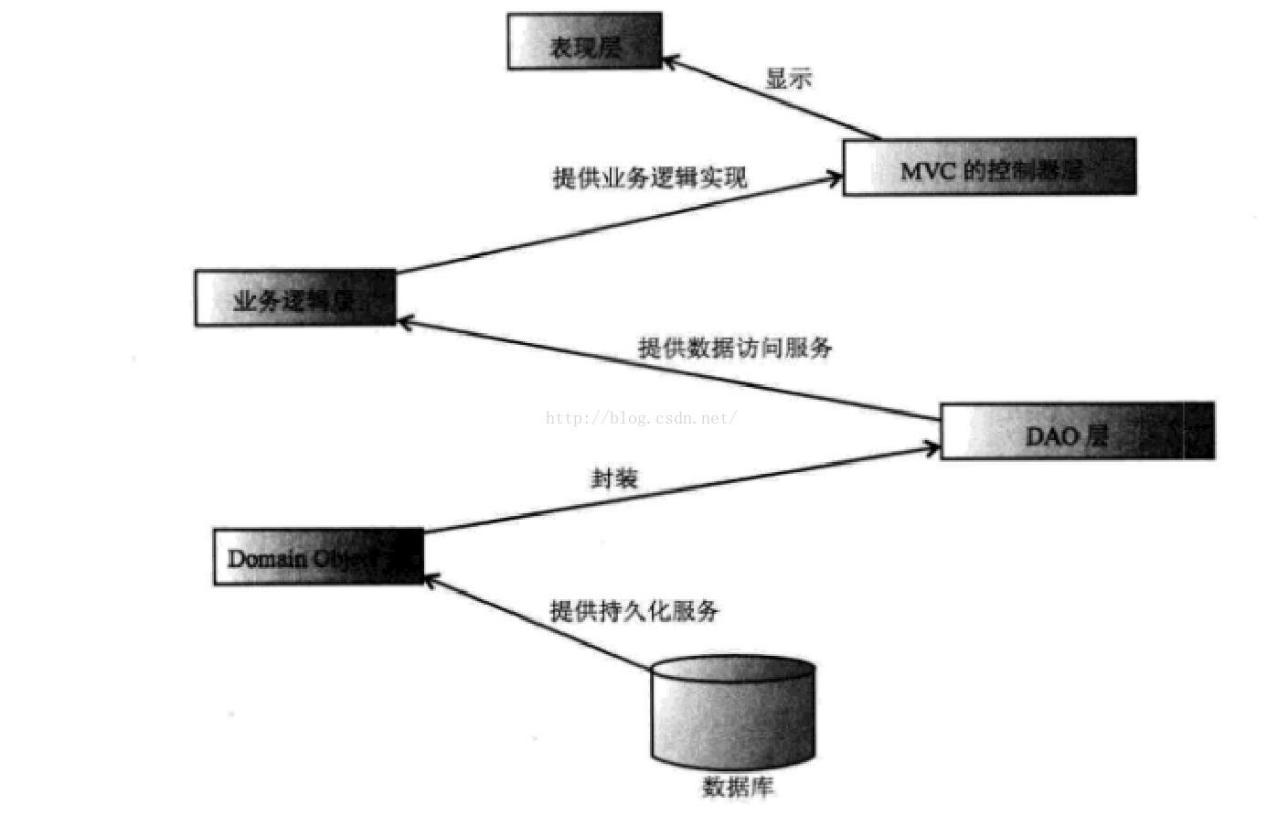
此层由一系列的业务逻辑对象组成，这些业务逻辑对象实现了系统所需要的业务逻辑方法。这些业务逻辑方法可能仅仅用于暴露Domain Object对象所实现的业务逻辑方法，也可能是依赖DAO组件实现的业务逻辑方法。

* Controller（控制器）层。

此层是由一系列控制器组成，这些控制器用于拦截用户的请求，并且调用业务逻辑组件的业务逻辑方法，处理用户请求，并且根据处理结果转发到不同的表现层组件。

* View（表现）层。

此层由一系列的JSP页面、Velocity页面、PDF文档视图组件组成，负责收集用户请求，并显示处理结果。



各层的JavaEE组件之间以松耦合的方式耦合在一起，各组件并不以硬编码的方式耦合，这种方式是为了应对以后的扩展需求。从上向下，上面组件的实现依赖于下面组件的功能；自下而上，下面组件支持上面组件的实现。

###### Java EE应用的组件

* 表现层组件

主要负责收集用户的输入的数据（比如表单），或者向用户显示系统状态。最常用的表现层技术是JSP，但JSP并不是唯一的表现层技术。表现层还可由Velocity、FreeMarker和Tapestry等技术完成，或者使用普通的应用程序充当表现层组件，甚至可以是小型智能设备。

* 控制器组件

对于Java EE的MVC框架而言，框架提供一个前端核心控制器，而核心控制器负责拦截用户请求，并将请求转发给用户实现控制器组件。而这些用户实现的控制器负责处理调用业务逻辑方法（控制器只起到switch的作用，具体的执行语句还是在case中的），处理用户请求。

* 业务逻辑组件

是的核心组件，实现系统的业务逻辑。通常，一个业务逻辑方法对应一次用户操作。一个业务逻辑方法应该是一个整体，因此要求对业务逻辑方法增加事务性（要干什么）。业务逻辑方法仅仅负责实现业务逻辑，不应该进行数据库的访问。以保证业务逻辑方法的实现，与具体的持久层访问技术分离。当系统需要在不同持久层技术之间切换时，系统的业务逻辑组件无需任何改变。

* DAO组件

也被称为数据访问对象。这个类型的对象比较缺乏变化（一般变得时候就是数据库的表又加了一个什么东东，这个最麻烦了），每个DAO组件都提供Domain Object对象基本的创建、查询、更新和删除等操作（在eclipse中有快捷键的哦），这些操作对应于数据表的CRUD（创建、查询、更新和删除）等原子操作（应该是指不可分割的最基本的操作）。当然，如果使用不同的持久层访问技术（数据库操作的相关技术，我猜的），DAO组件的实现会完全不同。为了业务逻辑组件的服务的实现与DAO组件的实现分离，程序应该为每个DAO组件都提供接口，业务逻辑组件面向DAO接口编程，这样才能提供更好的解耦。

* 领域对象组件

领域对象（Domain Object就是实体类啦）抽象了系统的对象模型，通常二页，这些实体类的状态都必须保存在数据库中。因此，每个实体类（领域对象）通常对应一个或多个数据表，实体类通常需要提供对数据记录访问方式。