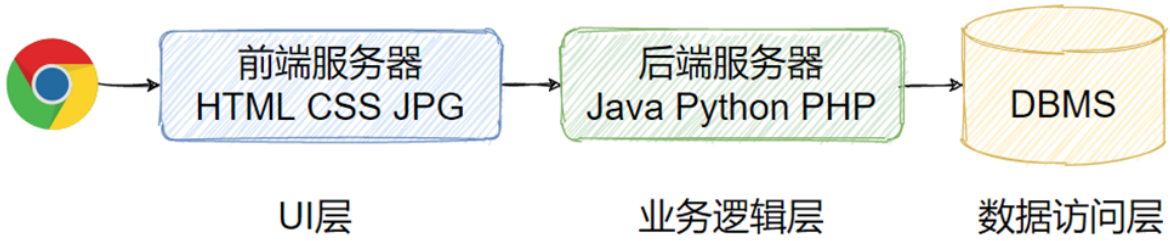
一、javaEE分层模型

web应用程序基本架构



不管是经典的JAVAEE架构，还是轻量级JavaEE架构，大致上都可以分为如下几层：

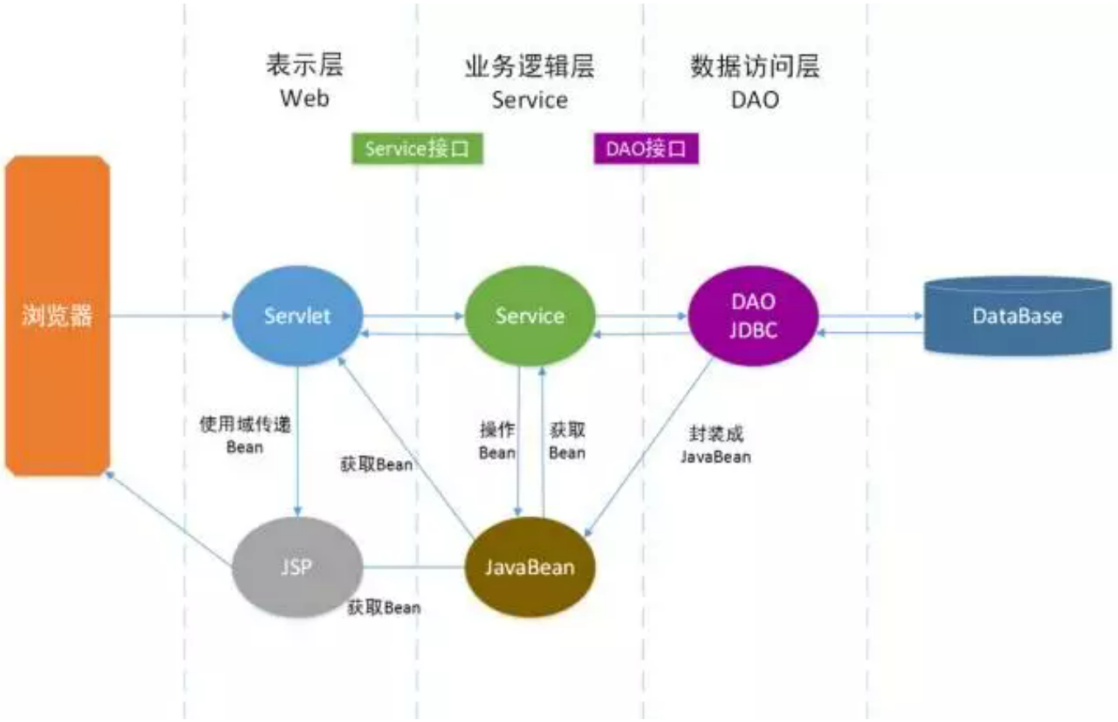
1、Domain Object（领域对象）层：此层由一系列的POJO（Plain Old Java Object）组成，这些对象是该系统的Domain Object,往往包含了各自所需实现的业务逻辑方法。

2、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（增加(Create)、读取(Retrieve)(重新得到数据)、更新(Update)和删除(Delete)））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层组件的作用基本类似。只是EAO层主要完成对实体（Entity）的CRUD操作，因此简称EAO层。}

3、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。

4、控制器层：次层由一系列控制器组成，这些控制器用于拦截用户请求，并调用业务逻辑组件的业务逻辑方法，处理用户请求，并根据处理结果转发到不同的表现层组件。

5、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。



三层架构

java项目中对应的结构

表示层：接受页面上的参数、找对应的sevice ， 执行对应的方法、跳转页面

逻辑层：接受表现层的参数，执行对应的业务逻辑（调用dao层方法），返回对应数据

持久层：就是对数据库做增删查。

二、Servlet

2.1 servlet 简介

Servlet是sun公司提供的一门用于开发动态web资源的技术。

Sun公司在其API中提供了一个servlet接口，用户若想用发一个动态web资源(即开发一个Java程序向浏览器输出数据)，需要完成以下2个步骤：

1、编写一个Java类，实现servlet接口。

2、把开发好的Java类部署到web服务器中。

按照一种约定俗成的称呼习惯，通常我们也把实现了servlet接口的java程序，称之为Servlet

2.2 Servlet 特点：

1）普通的java类，继承HttpServlet类，覆盖doGet、doPost等方法。

2）Servlet类只能交给tomcat服务器运行。

2.3、 怎样使用Eclipse开发Servlet?

1）编写一个servlet类，继承HttpServlet

```
public class Servlet extends HttpServlet{
        @Override
        protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOE
                System.out.println("This is Servlet!");
                resp.getWriter().write("This is Servlet!");
        }
}
```

2）配置web.xml文件

```
  <servlet>
        <servlet-name>Servlet</servlet-name>
        <servlet-class>zzuli.edu.cn.Servlet</servlet-class>
  </servlet>
  <servlet-mapping>
        <servlet-name>Servlet</servlet-name>
        <url-pattern>/Servlet</url-pattern>
  </servlet-mapping>
```

3）访问 http://localhost:8080/FirstServlet/Servlet

2.4、 在web.xml中配置 `<servlet>` 和 `<servlet-mapping>` 的作用?

servlet容器对url的匹配过程：

当一个请求发送到servlet容器的时候，容器先会将请求的url减去当前应用上下文的路径作为servlet的映射url，比如我访问的是http://localhost:8080/FirstServlet/Servlet，我的应用上下文是FirstServlet，容器会将http://localhost:8080/FirstServlet去掉，剩下的/Servlet部分拿来做servlet的映射匹配。

映射匹配步骤：

1）首先在web.xml文件中查找是否有匹配的url-pattern的内容（/Servlet）

2）如果找到匹配的url-pattern,则使用当前servlet-name的名称到web.xml文件中查询是否相同名称的servlet配置

3）如果找到相同名称的servlet配置，则取出对应的servlet配置信息中的servlet-class内容（zzuli.edu.cn.Servlet）,然后通过servlet-class里的内容，反射构造Servlet的对象，调用Servlet对象里面的方法。

2.5、Servlet 注解

Servlet3.0以上可以使用注解@WebServlet自动映射，不用在web.xml中配置 `<servlet>` 和 `<servlet-mapping>`

使用方法：编写一个servlet类，继承HttpServlet，然后在servlet上面加上@webServlet即可。

```
@WebServlet("/FirstServlet2")
public class FirstServlet2 extends HttpServlet{
        @Override
        protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOE
                System.out.println("This is Servlet2!");
                resp.getWriter().write("This is Servlet2!");
        }
}
```

2.6、九大内置对象

JSP中一共预先定义了9个这样的对象，分别为：request、response、session、application、out、pagecontext、config、page、exception

1.什么叫做内置对象?

答：在jsp中，有一些对象即开发者不需要自己去创建对象（全部由系统创建好），就可以直接使用对象调用相应的方法，这些由系统创建的对象称为内置对象。

2.九大内置对象分别是哪些?

内置对象名	类型	对象名	作用域
四大作用域对象:			
request	javax.servlet.http.HttpServletRequest	请求对象	Request
session	javax.servlet.http.HttpSession	会话对象	Session
application	javax.servlet.ServletContext	应用程序对象	Application
pageContext	javax.servlet.jsp.PageContext	页面上下文对象	Page
两个输出			
out	javax.servlet.jsp.JspWriter	输出对象	Page
response	javax.servlet.http.HttpServletResponse	响应对象	Page
三个打酱油			
page	java.lang.Object	页面对象	Page
config	javax.servlet.ServletConfig	配置对象	Page
exception	java.lang.Throwable	例外对象	Page

对作用域的解释:

Page域：只能在当前jsp页面使用（当前页面）。

Request域：只能在同一请求中使用（转发）。

Session域：只能在同一会话（session会话）中使用（私有的）。

Context（Application）域：只能在一个web应用中使用（全局）。

3.解析各个内置对象

3.1 request 对象

该对象代表客户端的请求信息，主要用于接收通过HTTP协议传送到服务器的数据（包括请求头信息，系统信息，请求方法以及请求参数等）。request对象的作用域为一次请求。

3.1.1 常用方法：

getParameter(String name) :返回指定参数名称的数值，如果没有相对应的数值则返回null。

getParameterValues(String name):返回具有相同参数名称的数值的集合，返回String类型的数组。

getRequestDispatcher(String uripath):页面的转发，地址不会发生改变，因为针对客户端来说只发送一次请求。

3.2 session 对象

session对象是由服务器自动创建的与用户请求相关的对象。服务器为每个用户都生成一个session对象，用于保存该用户的信息，跟踪用户的操作状态。session对象内部使用Map类来保存数据的，因此保存数据的格式为“key/value”。

session对象的value可以使用复杂的对象类型，不仅仅局限于字符串类型。

3.2.1 session对象就做会话：即每次浏览器访问网站，服务器就会给这个请求创建一个会话，存储到服务器端，服务器根据每一个会话的ID区别每一个请求的用户。

3.2.2 常用方法：

setAttribute(String key,Object obj):以key/value形式保存数据。

getAttribute(String key：通过key获取数据。

getId：获取session id。

invalidate():设置session对象失效。

setMaxInactiveInterval(int interval)：设置session对象的有效期。

removeAttribute(String key)：移除session中的属性。

3.2.3 Session过期时间的三种设置方式：

a. 在tomcat服务器的web.xml文件中进行设置，tomcat默认时间为30分钟。

web.xml文件在conf文件夹下，具体设置如下：（单位为分钟）

```
<session-config>

  <session-timeout>30</session-timeout>

</session-config>
```

b. 在项目中的web.xml文件中进行设置：

设置方法如a,优先顺序b>a,也就是说b设置了，a就无效了。

c. 在代码中设置：通过setMaxInactiveInterval()方法设置。

3.2.4 session 的销毁

a. 设置的时长到了以后自动销毁，常见的情况就是你在一个页面长时间不进行操作就要重新登录。

b. 调用invalidate()方法摧毁，常见的情况是注销登录。

3.2.5 注意事项

Session的存储类型可以是任意类型。存储位置在服务器端，安全性比较高。

在同一台电脑中，不同的浏览器也认为是不同的用户，也会分配一个session id。

session也会随着浏览器的关闭而失效。但请注意，session还会保留在服务器端，一直到设定的时间，才真正的被销毁。

3.2.5 cookie

cookie 是一个非常具体的东西，指的就是浏览器里面能永久存储的一种数据，仅仅是浏览器实现的一种数据存储功能。

cookie由服务器生成，发送给浏览器，浏览器把cookie以 K-V 形式保存到某个目录下的文本文件内，下一次请求同一网站时会把该cookie发送给服务器。由于cookie是存在客户端上的，所以浏览器加入了一些限制确保cookie不会被恶意使用，同时不会占据太多磁盘空间，所以每个域的cookie数量是有限的。

3.2.5 cookie 和session的关系

session的信息是通过sessionid获取的，而sessionid是存放在会话cookie当中的，当浏览器关闭的时候会话cookie消失，所以sessionid也就消失了，但是session的信息还存在服务器端，只是查不到所谓的session，但它并不是不存在。所以session在服务器关闭的时候，或者是session过期，又或者调用了invalidate()，再或者是session中的某一条数据消失调用session.removeAttribute()方法，session在通过调用session.getsession来创建的。

4、JDBC

JDBC 是sun公司提供的一套接口
接口都有调用者和实现者，面向接口调用，面向接口写实现类，都属于面向接口编程

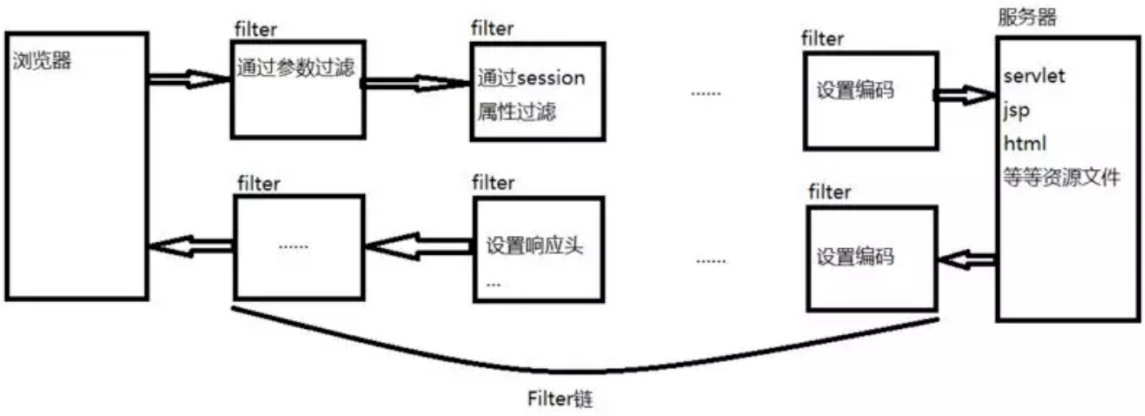
```
 public static final String URL = "jdbc:mysql://localhost:3306/imooc";
    public static final String USER = "liulx";
    public static final String PASSWORD = "123456";

    public static void main(String[] args) throws Exception {
        //1.加载驱动程序
        Class.forName("com.mysql.jdbc.Driver");
        //2. 获得数据库连接
        Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
        //3.操作数据库，实现增删改查
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT user_name, age FROM
imooc_goddess");
        //如果有数据，rs.next()返回true
        while(rs.next()){
            System.out.println(rs.getString("user_name")+" 年
龄："+rs.getInt("age"));
        }
    }
```

三、filter

过滤器是处于客户端与服务器资源文件之间的一道过滤网，在访问资源文件之前，通过一系列的过滤器对请求进行修改、判断等，把不符合规则的请求在中途拦截或修改。也可以对响应进行过滤，拦截或修改响应。

如下图，浏览器发出的请求先递交给第一个filter进行过滤，符合规则则放行，递交给filter链中的下一个过滤器进行过滤。过滤器在链中的顺序与它在web.xml中配置的顺序有关，配置在前的则位于链的前端。当请求通过了链中所有过滤器后就可以访问资源文件了，如果不能通过，则可能在中间某个过滤器中被处理掉。



过滤器一般用于登录权限验证、资源访问权限控制、敏感词汇过滤、字符编码转换等等操作，便于代码的重用，不必每个servlet中还要进行相应的操作。

过滤器的简单应用

1、新建一个class，实现接口Filter（注意：是javax.servlet中的Filter）。

2、重写过滤器的doFilter(request，response，chain)方法。另外两个init()、destroy()方法一般不需要重写。在doFilter方法中进行过滤操作。

3、在web.xml中配置过滤器。这里要谨记一条原则：在web.xml中，监听器>过滤器>servlet。也就是说web.xml中监听器配置在过滤器之前，过滤器配置在servlet之前，否则会出错。

```
<filter>
   <filter-name>loginFilter</filter-name>//过滤器名称
   <filter-class>com.nnngu.filter.loginFilter</filter-class>//过滤器类的包路径
   <init-param> //可选
       <param-name>参数名</param-name>//过滤器初始化参数
       <param-value>参数值</param-value>
   </init-pamm>
</filter>

<filter-mapping>//过滤器映射
   <filter-name>loginFilter</filter-name>
   <url-pattern>指定过滤器作用的范围</url-pattern>
</filter-mapping>
```

`<url-pattren>`处定义过滤器作用的范围。一般有以下规则：

1、作用与所有web资源：<url—pattern>/*`</url-pattern>`。则客户端请求访问任意资源文件时都要经过过滤器的过滤，通过则可以访问，否则不能访问。

2、作用于某一文件夹下所有文件：<url—pattern>/dir/*`</url-pattern>`

3、作用于某一种类型的文件：<url—pattern>.*扩展名* `</url-pattern>`。*比如<url—pattern>*.jsp `</url-pattern>`过滤所有对jsp文件的访问请求。

4、作用于某一文件夹下某一类型文件：<url—pattern>/dir/*.扩展名 `</url-pattern>`

如果一个过滤器需要过滤多种文件，则可以配置多个 `<filter-mapping>`，一个mapping定义一个url-pattern来定义过滤规则。如下：

```
<filter>
 <filter-name>loginFilter</filter-name>
 <filter-class>com.nnngu.filter.loginFilter</filter-class>
</filter>
<filter-mapping>
 <filter-name>loginFilter</filter-name>
 <url-pattern>*.jsp</url-pattern>
</filter-mapping>
<filter-mapping>
 <filter-name>loginFilter</filter-name>
 <url-pattern>*.do</url-pattern>
</filter-mapping>
```

例1：用过滤器实现登录验证，没登录则驳回访问请求并重定向到登录页面。

```
public void doFilter(ServletRequest arg0, ServletResponse arg1,
            FilterChain arg2) throws IOException, ServletException {

        HttpServletRequest request=(HttpServletRequest) arg0;
        HttpServletResponse response=(HttpServletResponse) arg1;
        HttpSession session=request.getSession();

        String path=request.getRequestURI();

        Integer uid=(Integer)session.getAttribute("userid");

        if(path.indexOf("/login.jsp")>-1){//登录页面不过滤
            arg2.doFilter(arg0, arg1);//递交给下一个过滤器
            return;
        }
        if(path.indexOf("/register.jsp")>-1){//注册页面不过滤
            arg2.doFilter(request, response);
            return;
        }

        if(uid!=null){//已经登录
            arg2.doFilter(request, response);//放行，递交给下一个过滤器

        }else{
            response.sendRedirect("login.jsp");
        }

}
```

四、反射

JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。Java的反射机制是Java特性之一，反射机制是构建框架技术的基础所在。

```
public class Apple {
```

```
    private int price;

    public int getPrice() {
        return price;
    }

    public void setPrice(int price) {
        this.price = price;
    }

    public static void main(String[] args) throws Exception{
        //正常的调用
        Apple apple = new Apple();
        apple.setPrice(5);
        System.out.println("Apple Price:" + apple.getPrice());
        //使用反射调用
        Class clz = Class.forName("com.chenshuyi.api.Apple");
        Method setPriceMethod = clz.getMethod("setPrice", int.class);
        Constructor appleConstructor = clz.getConstructor();
        Object appleObj = appleConstructor.newInstance();
        setPriceMethod.invoke(appleObj, 14);
        Method getPriceMethod = clz.getMethod("getPrice");
        System.out.println("Apple Price:" + getPriceMethod.invoke(appleObj));
    }
}
```

1、加载类信息

Class clz= Class.forName("com.mashibing.test.Apple");

2、获得对象

Constructor appleConstructor = clz.getConstructor();

Object appleObj = appleConstructor.newInstance();

3、获得方法

Method setPriceMethod = clz.getMethod("setPrice", int.class);

4、执行方法

setPriceMethod.invoke(appleObj, 14);

五、SSM框架

SSM到底是什么。我们照搬百度百科的解释：**SSM（Spring+SpringMVC+MyBatis）框架集由Spring、MyBatis两个开源框架整合而成（SpringMVC是Spring中的部分内容）。常作为数据源较简单的web项目的框架。**我们可以发现，ssm其实就是Spring，SpirngMVC，Mybatis的缩写。那么问题来了，SSM分别是干什么的呢？要分析这个问题，就不得不搬出我们javaweb传统开发的MVC框架了。

maven：

一、Maven简介

Maven可译为“知识的积累”、“专家”，主要服务于基于Java平台的项目构建、依赖管理和项目信息管理。

1、Maven—项目构建工具

帮助我们自动化构建过程，从清理、编译、测试到生成报告，再到打包部署。我们需要做的是使用Maven配置好项目，然后输入简单的命令，Maven会帮我们处理上上述任务。譬如我们进行测试，只需要遵循Maven的约定编写好测试用例，当我们运行构建的时候，这些测试便会自动进行。

Maven抽象了一个完整的构建生命周期模型，模型吸取了大量的其他构建脚本和工具的优点，总结了大量项目的实际需求，Maven还帮我们标准化构建过程。

2、Maven—项目依赖管理和项目信息管理工具

在这个开源时代，几乎任何java应用都会借用一些第三方的开源类库，这些类库都可以通过依赖的方式引入到项目中来。随着依赖的增多，版本不一致、冲突，依赖问题接踵而至。maven提供了一个优秀的解决方案，通过一个坐标系统准确的定位每一个构件(artifact)，也就是通过一组坐标maven能够找到任何一个Java类库（如jar文件）。

maven还帮我们管理各个项目信息，项目描述、开发者列表、版本控制系统地址、许可证、缺陷管理系统地址等。maven还为Java开发者提供了一个免费的中央仓库，几乎可以找到任何的流行开源类库，帮我们自动下载构建。

下载jar包：maven项目会有一个pom.xml文件，在这个文件中，添加相应配置，maven会自动帮我们下载相应jar包。`<dependencies>`结点里，每配置一个`<dependency>`

`<groupId>`org.springframework`</groupId>`项目名

`<artifactId>`spring-webmvc`</artifactId>`项目模块`<version>`3.0.5.RELEASE`</version>`项目版本

maven都会通过：项目名-项目模块-项目版本，在互联网上的代码库中下载相应jar包。

下载依赖：在maven的代码库中，每一个jar包也有自己的pom.xml文件，而这个文件里面也会有`<dependency>`配置，配置的jar包所依赖的其他jar包都会被maven自动下载下来。

五、JDBC

Java数据库连接，（Java Database Connectivity，简称JDBC）是Java语言中用来规范客户端程序如何来访问数据库的应用程序接口，提供了诸如查询和更新数据库中数据的方法。JDBC也是Sun Microsystems的商标。我们通常说的JDBC是面向关系型数据库的。

```
public static final String URL = "jdbc:mysql://localhost:3306/imooc";
    public static final String USER = "liulx";
    public static final String PASSWORD = "123456";

    public static void main(String[] args) throws Exception {
        //1.加载驱动程序
        Class.forName("com.mysql.jdbc.Driver");
        //2. 获得数据库连接
        Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);
        //3.操作数据库，实现增删改查
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery("SELECT user_name, age FROM
imooc_goddess");
        //如果有数据，rs.next()返回true
        while(rs.next()){
            System.out.println(rs.getString("user_name")+" 年
龄："+rs.getInt("age"));
        }
    }
```