SSHM

重点是 SSM
Struts2,hibernate(了解)

1. Struts2

1) 描述下如何使用 SSH 进行整合开发

SSH 框架的系统从职责上分为四层：表示层、业务逻辑层、数据持久层和域模块层，帮助开发人员在短期内搭建 Web 应用程序。其中使用 Struts 作为系统的整体基础架构，负责 MVC 的分离，控制业务跳转，利用 Hibernate 框架对持久层提供支持，Spring 做管理，管理 struts 和 hibernate。

a、Struts

Struts 是个表示层框架，负责界面展示，接收请求，分发请求。

b、Hibernate

Hibernate 是个持久层框架，负责与数据库的交互。

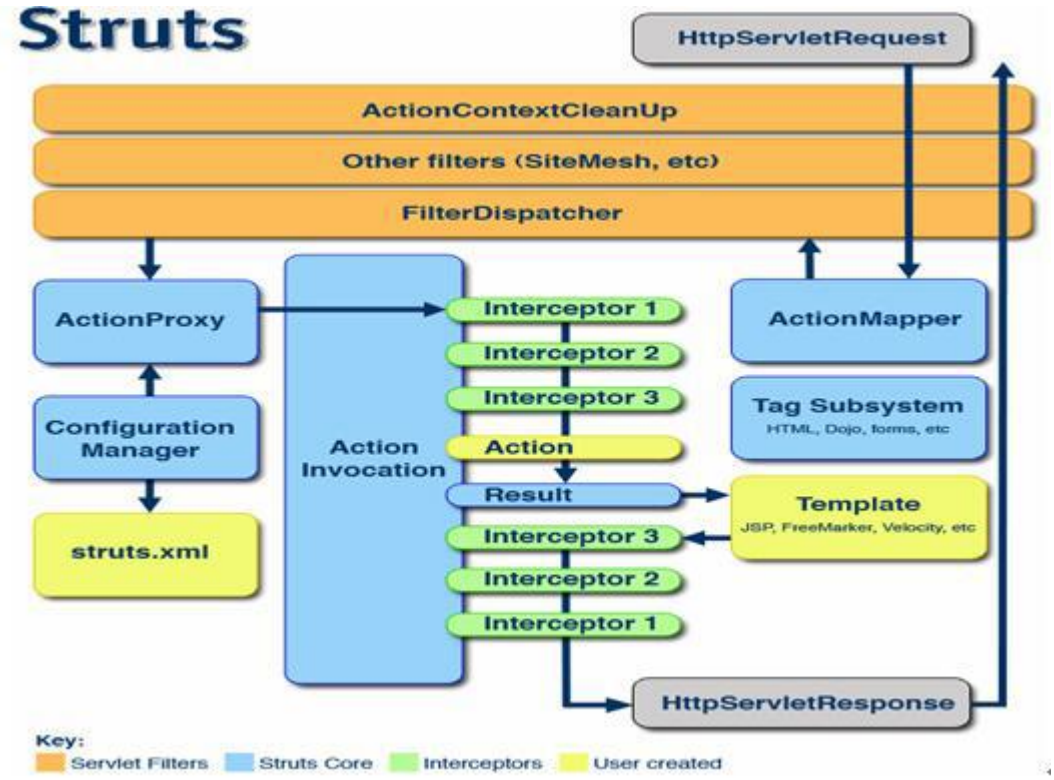
c、Spring

Spring 是一个业务层框架，也是一个整合框架，管理 struts 和 hibernate，提供 IOC 和 AOP 的功能。

Action Service Dao entity（表）

2) 描述 struts2 的工作原理

StrutsPrepareAndExecuteFilter



Request--->Filter--->ActionProxy-->
Before Invoke action(Interceptor stack 1,2,3)
Action-->String-->Result-->jsp,freemarker
After invoke action(Interceptor stack 3,2,1)
Response

拦截器

```
public class UserAction{
    private User user;
    private File file;
}
```

Struts 默认提供的拦截器
1，属性注入（数据类型转换）
2，校验
3，文件上传
4，国际化

自定义拦截器
5，权限控制

重点：Filter,拦截器
首先，struts2 会在 web.xml 配置一个核心的过滤器，拦截 action 的请求
其次，会先经过一系列的拦截器，然后再到 action

Action 处理之后，会返回对应的 result
之后，会再一次经过之前拦截器，然后才响应到客户端

3) Struts2 的工作模式是线程安全还是非安全？

安全的，因为每次来一个请求就会创建一个 action 的实例

4) Struts2 如何实现国际化？

配置国际化资源文件，做几个国家的国际化，就写几个配置文件
配置文件里面就写
key=value
username=username
username=用户名
程序中，通过 key 获取对应的值

2. Hibernate

1) 谈谈对 ORM 的理解

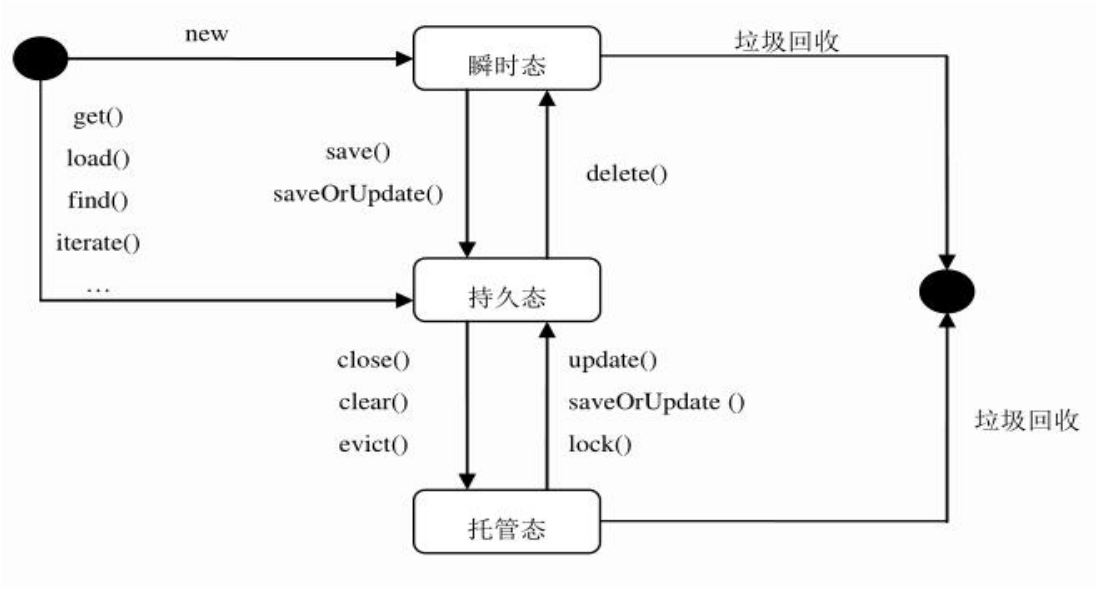
ORMapping
对象关系映射，对象　映射 表的记录
一个对象就对应着表的一条记录
好处：（用它和不用它区别）

```
Save(Student stu)
1，insert into student(name,age) values(?,?)
ps.setString(1,stu.getName())
//映射关系要有程序员自己来做

2，getList();
While(rs.next()){
    Student s = new Student();
    s.setName(rs.getString("name"));
    ....
}
```

ORM:对象关系映射

2) 谈谈 Hibernate 对象的三种状态



瞬时状态：transient，session 没有缓存对象，数据库也没有对应记录，OID 没有值

持久状态：persistent，session 有缓存对象，数据库有对应的记录，OID 有值

脱管状态：detached，session 没有缓存对象，数据库也有记录，OID 有值

3) getCurrentSession(),openSession()的区别

getCurrentSession():如果当前已经有 session，就用当前的，否则会创建一个新的

OpenSession():每次都创建新的

事务控制：getCurrentSession()

4) Casecade 和 inverse 配置的作用

Casecade:级联，级联删除，级联保存
Inverse:反转，双方的维护关系由谁来维护的问题
一对多的情况，这个时候将关系交给多的一方来维护
在一的一方设置 inverse=true
如果让一的一方来维护，会额外发送多余的 SQL 语句（问题）
Order
Inverse=true
OrderItem

5) 谈谈对一级缓存和二级缓存的理解

一级缓存：session 级别的缓存
二级缓存：sessionFactory 级别的缓存
推荐的商品信息--->sessionFactory
查询缓存

```
<!-- 开启二级缓存，默认是关闭 -->
            <property name="hibernate.cache.use_second_level_cache">true</property>
            <!-- 我们采用缓存提供商是哪个 -->
            <property name="hibernate.cache.provider_class">org.hibernate.cache.EhCacheProvider</property>
            <!-- 打开查询缓存 -->
            <property name="hibernate.cache.use_query_cache">true</property>
<!-- 要缓存的类是什么 -->
            <class-cache usage="read-write" class="com.qianfeng.entity.Customer"/>
            <class-cache usage="read-only" class="com.qianfeng.entity.OrderInfo"/>
            <collection-cache usage="read-only" collection="com.qianfeng.entity.Customer.orderSet"/>
```

代码里面：

```
Query query = session.createQuery("from Customer where id=:id");
query.setInteger("id", 3);
//关键
query.setCacheable(true);
```

--->缓存服务器 redis

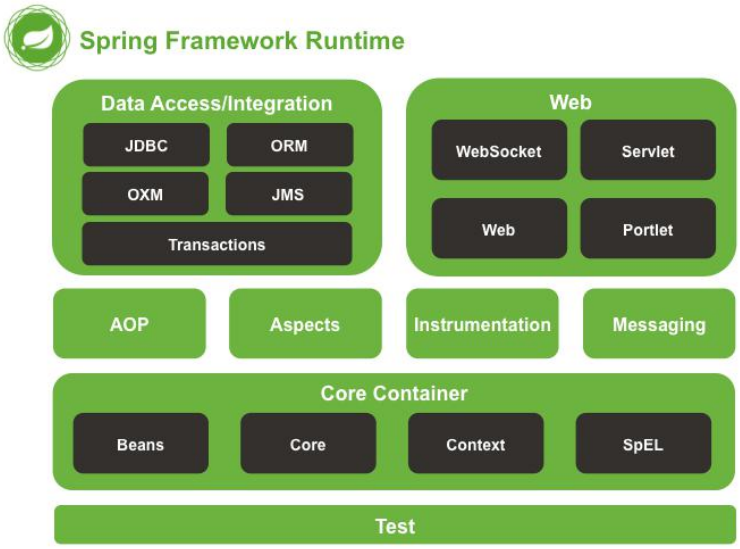
6) 描述什么是 OpenSessionInview 的机制

Cutomer 1 -- n Order
Customer
Cutomer orderSet
懒加载异常
1，lazy=false
2，按需加载 渲染页面的时候才能确定 Filter session close

实现的关键技术是 Filter
解决的问题是：懒加载问题
当你渲染页面的时候，如果需要使用到关联的数据，由于 session 已经关闭了，所以将会报懒加载异常
但是，如果不配置懒加载，就是立即加载，在有些应用场景用不上关联的数据，就会造成浪费
期望的目标：按需获取
解决的思路：上述的情况，核心问题就是渲染页面的时候，session 已经关闭
先渲染页面，再关闭 session
用 Filter 来控制 doFilter（session 的开启和关闭）

3. Spring

1) 谈谈你对 spring 的认识



1.方便解耦，简化开发
通过 Spring 提供的 IoC 容器，可以将对象之间的依赖关系交由 Spring 进行控制，避免硬编码所造成的过度程序耦合。

Controller

@Autowired

private UserService userService;

2.AOP 编程的支持
通过 Spring 提供的 AOP 功能，方便进行面向切面的编程，如性能监测、事务管理、日志记录等。

3.声明式事务的支持
4.方便集成各种优秀框架

5.降低 Java EE API 的使用难度
如对 JDBC，JavaMail，远程调用等提供了简便封封装

2) 什么是 IOC 和 DI

IOC 控制反转：说的是创建对象实例的控制权从代码控制剥离到 IOC 容器控制，侧重于原理。

DI 依赖注入：说的是创建对象实例时，为这个对象注入属性值或其它对象实例，侧重于实现。

控制反转是一种思想，如何实现控制反转呢，注入是一种实现手段。

在 Java 中依然注入有以下三种实现方式：

1. 构造器注入
2. Setter 方法注入
3. 接口注入

Spring 中 BeanFactory 接口是 Spring IoC 容器的核心接口。ApplicationContex 接口对 BeanFactory（是一个子接口）进行了扩展.我们常用的 **ClassPathXmlApplicationContext**

Web.xml 监听器来负责启动初始化 spring 容器

3) Spring 有几种配置方式

基于 XML 和基于注解，管理依赖用注解，事务用 XML

1，配置数据源
2，配置事务管理器，引用数据源
3，配置事务策略，引用事务管理器
4，配置事务的 AOP，设置哪些类的哪些方法需要应用这个事务策略

4) Spring Bean 的作用域有哪些？

singleton 这种 bean 范围是默认的，这种范围确保不管接受到多少个请求，每个容器中只有一个 bean 的实例，单例的模式由 bean factory 自身来维护。
prototype 原形范围与单例范围相反，为每一个 bean 请求提供一个实例

实际的应用场景：
Service,dao---单例
Struts2 action---多例 prototype

5) Spring 中的单例 Bean 是线程安全的吗？

Spring 框架并没有对单例 bean 进行任何多线程的封装处理。关于单例 bean 的线程安全和并发问题需要开发者自行去搞定。但实际上，大部分的 Spring bean 并没有可变的状态(比如 Service 类和 DAO 类)，所以在某种程度上说 Spring 的单例 bean 是线程安全的。如果你的 bean 有多种状态的话（比如 View Model 对象），就需要自行保证线程安全。

有状态的 bean，无状态的 bean

6) 什么是 bean 的装配

装配，或 bean 装配是指在 Spring 容器中把 bean 组装到一起，前提是容器需要知道 bean 的依赖关系，如何通过依赖注入来把它们装配到一起。

例子：
就是把 dao 对象注入给 service 对象，这个过程就叫装配

7) Spring 的自动装配机制

在 Spring 框架中，在配置文件中设定 bean 的依赖关系是一个很好的机制，Spring 容器还可以自动装配合作关系 bean 之间的关联关系。
一种是在 XML 的配置文件中，
<bean id="userBiz" class="com.qianfeng.biz.impl.UserBizImpl" autowire="byName"/>
加上 autowire="byName" 或 autowire="byType"

一种是注解的方式，<context:annotation-config />，然后在类中用@Autowired 即可

```
@Controller
Public class UserController{
    @AutoWried
    private IUserService userService;
}
@Service
Public class UserServiceImpl implements IUserService{
      @Autowried
      Private IUserDao userDao;
}
```

8) 自动装配，byName 和 byType 的区别

byName：该选项可以根据 bean 名称设置依赖关系。当向一个 bean 中自动装配一个属性时，容器将根据 bean 的名称自动在在配置文件中查询一个匹配的 bean。如果找到的话，就装配这个属性，如果没找到的话就报错。
byType：该选项可以根据 bean 类型设置依赖关系。当向一个 bean 中自动装配一个属性时，容器将根据 bean 的类型自动在在配置文件中查询一个匹配的 bean。如果找到的话，就装配这个属性，如果没找到的话就报错。

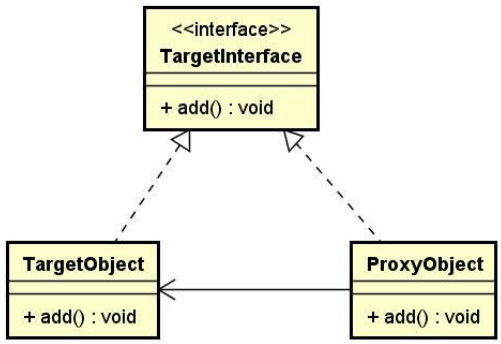
9) Spring 常用的注解

@Autowired，@Resource 自动装配依赖，默认是按照类型来装配
@Resource(name="userDao")，自动按照名字来装配
@Scope("prototype")
@Component(各个层都可以用这个注解，不过建议用下面的，可读性更高)
@Repository：dao 层
@Service：service 层
@Controller：web 层

10) 描述下代理模式的类结构

代理模式是一种设计模式
静态代理和动态代理是一种创建代理对象的实现方式

代理是通知目标对象后创建的对象。从客户端的角度看，代理对象和目标对象是一样的。




```
Public class ProxyObject{
        Private TragetObject target;

        Public void add(){
            //before add
            target.add();
            //after add
        }
}
```

11) 谈谈静态代理和动态代理的区别

静态代理：由程序员创建或特定工具自动生成源代码，再对其编译。在程序运行前，代理类的.class 文件就已经存在了。

动态代理：在程序运行时，运用反射机制动态创建而成。
两种的区别，在于创建代理对象方式上的区别。

12) 动态代理的两种实现方式：JDK 动态代理和 CGLIB 的区别

基于 JDK 的动态代理，接口
基于 CGlib 的动态代理，类
CGlib 采用非常底层的字节码技术，可以为一个类创建子类，并在子类中采用方法拦截技术拦截所有父类方法的调用，并织入横切逻辑
Spring 会自动在这两种方式进行切换
创建代理的时候，jdk 的方式快
使用的时候，cglib 的方式快

13) 谈谈你对 AOP 的理解

AOP：面向切面编程，可以将核心的业务逻辑和横切逻辑做分离
比如，我们可以用 AOP 加入一些日志记录，加入性能检测，还有，事务的控制，还有读写分离的控制

不改变原有的业务代码，动态增加新的横向处理逻辑
比如，事务控制，性能检测，读写分离

14) 如何配置声明式的事务控制

```
<!-- 配置数据 DBCP 源 -->
     <bean id="datasource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
          <property name="driverClass" value="${driverClass}"></property>
          <property name="jdbcUrl" value="${url}"></property>
          <property name="user" value="${user}"></property>
          <property name="password"  value="${password}"></property>
     </bean>

<!-- 配置 JDBC 的事务管理器 -->
     <bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
          <property name="dataSource" ref="datasource"/>
     </bean>

<!-- 配置事务策略 -->
     <tx:advice id="txAdvice" transaction-manager="txManager">
          <tx:attributes>
```

```
            <tx:method name="*" propagation="REQUIRED"
                 isolation="DEFAULT" />
            <tx:method name="get*" read-only="true"/>
        </tx:attributes>
    </tx:advice>
    <!-- 配置事务策略的应用范围 -->
    <aop:config>
        <aop:advisor advice-ref="txAdvice"
            pointcut="execution(* com.qianfeng.service..*.*(..))" />
    </aop:config>
```

1，配置数据源

2，配置事务管理器

3，配置事务策略，哪些需要加事务，哪些只是只读事务

4，配置事务的应用范围，哪些类哪些方法需要加事务的控制，同时关联上事务策略

15) spring 如何整合 hibernate

配置数据源 datasource，

配置 sessionFactory，

配置 HibernateTemplate 支持 HibernateDaoSupport 开发，简化 dao 层开发

UserDao extends HibernateDaoSupport

```
<bean id="datasource" class="com.mchange.v2.c3p0.ComboPooledDataSource">
        <property name="driverClass" value="${driverClass}"></property>
        <property name="jdbcUrl" value="${url}"></property>
        <property name="user" value="${user}"></property>
        <property name="password"  value="${password}"></property>
    </bean>

   <!-- 加载整合 hibernate 配置文件，并以此生成 SessionFactory 对象 -->
   <bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
     <property name="dataSource" ref="datasource"/>
     <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
            <prop key="hibernate.show_sql">true</prop>
            <prop key="hibernate.format_sql">true</prop>
            <prop key="hibernate.hbm2ddl.auto">update</prop>
        </props>
     </property>
     <property name="mappingLocations" value="classpath:com/qianfeng/entity/*.hbm.xml"></property>
   </bean>
```

```xml
<bean id="accountDao" class="com.qianfeng.dao.impl.AccountDaoImpl">
        <property name="sessionFactory" ref="sessionFactory"/>
    </bean>
```

```java
public class AccountDaoImpl extends HibernateDaoSupport implements IAccountDao {

    @SuppressWarnings("all")
    @Override
    public void in(final String account, final double money) {
        //如果是普通的save，一句话搞定
        //this.getHibernateTemplate().save(null);

        this.getHibernateTemplate().execute(new HibernateCallback() {
            @Override
            public Object doInHibernate(Session session) throws HibernateException, SQLException {
                Query query =
                        session.createQuery("update Account a set a.balance=a.balance-:money where a.account=:account");
                query.setDouble("money", money);
                query.setString("account", account);
                return query.executeUpdate();
            }
        });
    }
```

16) spring 如何整合 struts2

Web.xml 配置监听器，启动 spring 容器
通过 spring 容器为 struts 的 action 注入 service 对象，也可以将 action 交给 spring 来管理，但是其 scope 类型要为 prototype

17) 描述 spring 里面涉及到的设计模式

代理模式：AOP
单例模式：spring 的配置文件中定义的 bean 默认为单例模式
模板方法：用来解决代码重复的问题，比如 jdbcTemplate

```java
public class UserDao extends JdbcDaoSupport{

    public void add(){

        this.getJdbcTemplate().update("insert into user(username,password) values(?,?)", "1605","123456");

    }

}
```

工厂模式：BeanFactory 用来创建对象的实例

18) 请描述 springMVC 的工作原理

核心控制 DispatcherServlet
业务控制器 Controller
返回一个 String 表示逻辑视图---视图解析器---物理视图----PC
返回一个 json 的数据，@ResponseBody String,User,List<User>----PC,APP
<user>
<username>zhangsan</username>

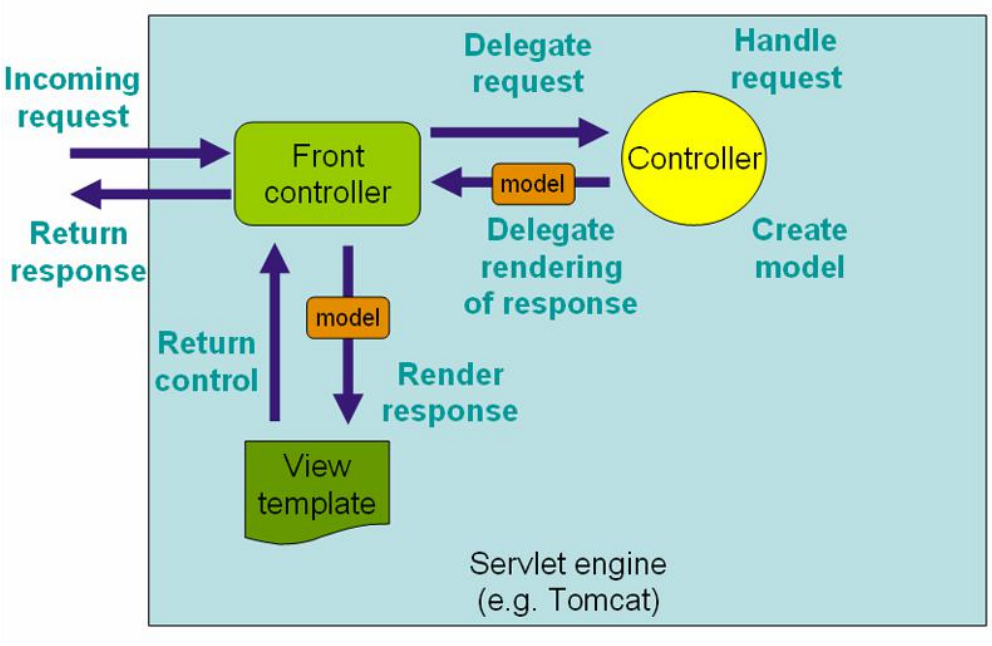
</user>

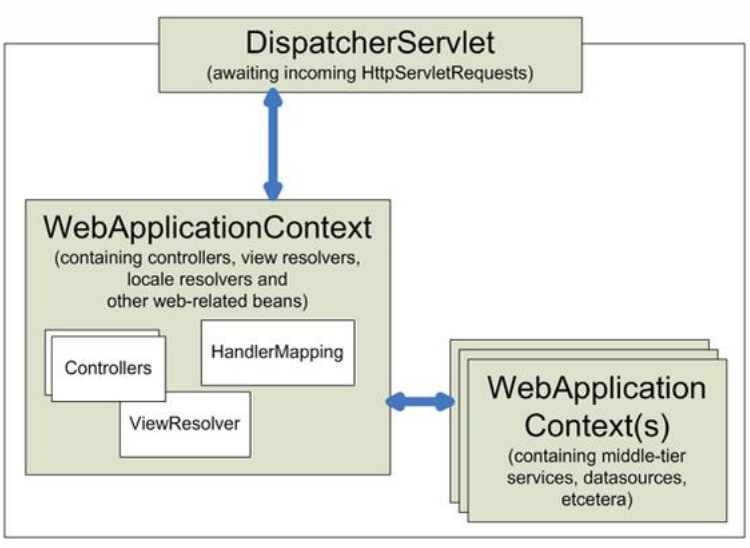
{username:”zhangsan”}

[{username:”zhangsan”},{username:”lisi”}]

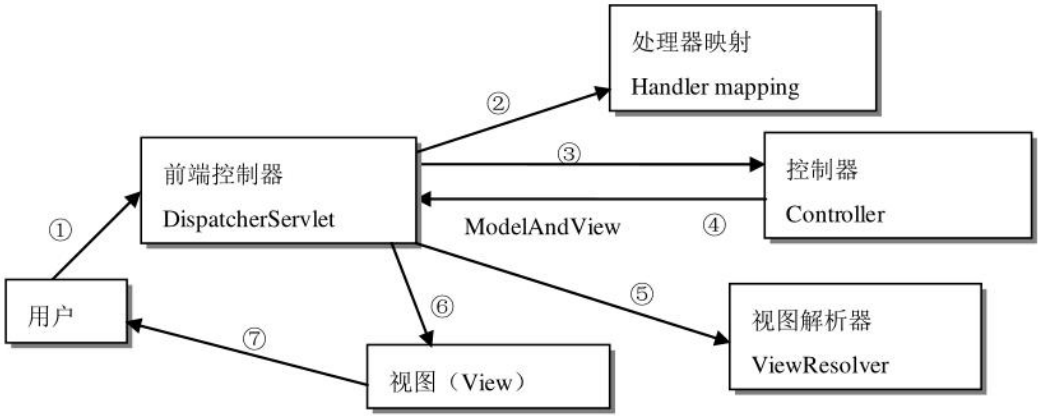
业务对象 Service

数据层对象 Dao/Mapper





Context hierarchy in Spring Web MVC



@RequestMapping()

首先，会由前端控制器 DispatcherServlet 来进行拦截，
然后结合 HandlerMappering 找到对应的 Controller，
Controller 经过处理之后，会返回一个 ModelAndView 对象，
但是 view 是逻辑视图，所以还需要借助 ViewResolver 来返回对应的物理视图，
最后物理视图结合模型数据进行渲染响应给客户端

19) DispatcherServlet

Spring 的 MVC 框架是围绕 DispatcherServlet 来设计的，它用来处理所有的 HTTP 请求和响应。

20) 常用注解

@Controller
@RequestMapping
@ResponseBody
@PathVariable

@RequestMapping(“user/{userId}”)
Public User getById(@PathVariable long userId){
}

4. Mybatis

21) Mybatis vs Hibernate

MyBatis 是一个半自动的映射框架，之所以称为半自动，是因为它需要手工匹配提供的 POJO、SQL、和映射关系，而 Hibernate 只需提供 POJO 和映射关系即可。

由于 Mybatis 需要自己来编写 SQL，所以性能优化会更加方便
Hibernate 也有提供方法来执行原生的 SQL,他的其他操作比如 HQL 都是对 SQL 操作做了一层的封装

Mapper 的映射文件去配置映射关系，insert select update 对应的映射语句
Save(user) hibernate 不需要为这样的操作写映射

22) Mybatis 开发的流程

接口：UserMapper
映射文件：UserMapper.xml
不需要写实现，实现由 mybatis 帮我们实现了，代理的模式。
实际开发中，可以用自动的方式来生成对应的 entity，Mapper 及映射文件

23) Spring 整合 mybatis（跟 hibernate 讲述差别）

```
1.配置数据源
2<!-- 4,整合mybatis，配置SqlSessionFactory -->
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />
    <property name="configLocation" value="classpath:mybatis-config.xml" />
    <property name="mapperLocations" value="classpath:com/dream/ssm/mapper/*.xml" />
</bean>
3
<!-- 5.3，配置JDBC的事务管理器 -->
<bean id="txManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
    <property name="dataSource" ref="dataSource"/>
</bean>
```