SpringBoot-SSM复习

Spring框架

Spring是什么

Spring是一个轻量级的IoC和AOP容器框架。是为Java应用程序提供基础性服务的一套框架，目的是用于简化企业应用程序的开发，它使得开发者只需要关心业务需求。

常见的配置方式有三种：基于XML的配置、基于注解的配置、基于Java的配置。

主要由以下几个模块组成：

- Spring Core：核心类库，提供IoC/DI容器
- Spring Context：提供框架式的Bean访问方式，以及企业级功能（JNDI、定时任务等）
- Spring AOP：支持AOP编程支持
- Spring DAO：对JDBC的抽象，简化了数据访问异常的处理
- Spring ORM：对现有的ORM框架的支持，例如JPA-Hibernate
- Spring Web：提供了基本的面向Web的综合特性，例如多方文件上传
- Spring MVC：提供面向Web应用的Model-View-Controller实现

Spring 的优点？

- spring属于低侵入式设计，代码的污染极低
- spring的DI机制将对象之间的依赖关系交由框架处理，减低组件的耦合性
- Spring提供了AOP技术，支持将一些通用任务，如安全、事务、日志、权限等进行集中式管理，从而提供更好的复用
- spring对于主流的应用框架提供了集成支持

Spring的AOP理解

OOP面向对象，允许开发者定义纵向的关系，但并适用于定义横向的关系，导致了大量代码的重复，而不利于各个模块的重用 AOP一般称为面向切面，作为面向对象的一种补充，用于将那些与业务无关，但却对多个对象产生影响的公共行为和逻辑，抽取并封装为一个可重用的模块，这个模块被命名为切面Aspect，减少系统中的重复代码，降低了模块间的耦合度，同时提高了系统的可维护性。可用于权限认证、日志、事务处理。 AOP实现的关键在于代理模式，AOP代理主要分为静态代理和动态代理。静态代理的代表为AspectJ；动态代理则以Spring AOP为代表

> AspectJ是静态代理的增强，所谓静态代理，就是AOP框架会在编译阶段生成AOP代理类，因此也称为编译时增强，他会在编译阶段将AspectJ(切面)织入到Java字节码中，运行的时候就是增强之后的AOP对象 Spring AOP使用的动态代理，所谓的动态代理就是说AOP框架不会去修改字节码，而是每次运行时在内存中临时为方法生成一个AOP对象，这个AOP对象包含了目标对象的全部方法，并且在特定的切点做了增强处理，并回调原对象的方法

Spring AOP中的动态代理主要有两种方式，JDK动态代理和CGLIB动态代理

> JDK动态代理只提供接口的代理，不支持类的代理。核心InvocationHandler接口和Proxy类，InvocationHandler 通过invoke()方法反射来调用目标类中的代码，动态地将横切逻辑和业务编织在一起；接着，Proxy利用 InvocationHandler动态创建一个符合某一接口的的实例，生成目标类的代理对象

> 如果代理类没有实现 InvocationHandler 接口，那么Spring AOP会选择使用CGLIB来动态代理目标类。CGLIB（Code Generation Library），是一个代码生成的类库，可以在运行时动态的生成指定类的一个子类对象，并覆盖其中特定方法并添加增强代码，从而实现AOP。CGLIB是通过继承的方式做的动态代理，因此如果某个类被标记为final，那么它是无法使用CGLIB做动态代理的
>
> 静态代理与动态代理区别在于生成AOP代理对象的时机不同，相对来说AspectJ的静态代理方式具有更好的性能，但是AspectJ需要特定的编译器进行处理，而Spring AOP则无需特定的编译器处理。

Spring的IoC理解

1、IoC就是控制反转，是指创建对象的控制权的转移，以前创建对象的主动权和时机是由自己把控的，而现在这种权力转移到Spring容器中，并由容器根据配置文件去创建实例和管理各个实例之间的依赖关系，对象与对象之间松散耦合，也利于功能的复用。DI依赖注入，和控制反转是同一个概念的不同角度的描述，即 应用程序在运行时依赖IoC容器来动态注入对象需要的外部资源。 2、最直观的表达就是IOC让对象的创建不用去new了，可以由spring自动生产，使用java的反射机制，根据配置文件在运行时动态的去创建对象以及管理对象，并调用对象的方法的。 3、Spring的DI有三种注入方式 ：构造器注入、setter方法注入、根据注解注入。 IoC让相互协作的组件保持松散的耦合，而AOP编程允许你把遍布于应用各层的功能分离出来形成可重用的功能组件。

解释Spring支持的几种bean的作用域

Spring容器中的bean可以分为5个范围： 1、singleton默认，每个容器中只有一个bean的实例，单例的模式由BeanFactory自身来维护 2、prototype：为每一个bean请求提供一个实例 3、request：为每一个网络请求创建一个实例，在请求完成以后，bean会失效并被垃圾回收器回收 4、session：与request范围类似，确保每个session中有一个bean的实例，在session过期后，bean会随之失效 5、global-session：全局作用域，global-session和Portlet应用相关。当你的应用部署在Portlet容器中工作时，它包含很多portlet。如果你想要声明让所有的portlet共用全局的存储变量的话，那么这全局变量需要存储在global-session中。全局作用域与Servlet中的session作用域效果相同。

Spring框架中的单例Beans是线程安全的么

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Serview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。最浅显的解决办法就是将多态bean的作用域由“singleton”变更为“prototype”

Spring如何处理线程并发问题

- 在一般情况下，只有无状态的Bean才可以在多线程环境下共享，在Spring中，绝大部分Bean都可以声明为singleton作用域，因为Spring对一些Bean中非线程安全状态采用ThreadLocal进行处理，解决线程安全问题。
- ThreadLocal和线程同步机制都是为了解决多线程中相同变量的访问冲突问题。同步机制采用了“时间换空间”的方式，仅提供一份变量，不同的线程在访问前需要获取锁，没获得锁的线程则需要排队。而ThreadLocal采用了“空间换时间”的方式
- ThreadLocal会为每一个线程提供一个独立的变量副本，从而隔离了多个线程对数据的访问冲突。因为每一个线程都拥有自己的变量副本，从而也就没有必要对该变量进行同步了。ThreadLocal提供了线程安全的共享对象，在编写多线程代码时，可以把不安全的变量封装进ThreadLocal

Spring的自动装配

在spring中，对象无需自己查找或创建与其关联的其他对象，由容器负责把需要相互协作的对象引用赋予各个对象，使用autowire来配置自动装载模式。 在Spring框架xml配置中共有5种自动装配：

- no：默认的方式是不进行自动装配的，通过手工设置ref属性来进行装配bean
- byName：通过bean的名称进行自动装配，如果一个bean的 property 与另一bean 的name 相同，就进行自动装配
- byType：通过参数的数据类型进行自动装配。
- constructor：利用构造函数进行装配，并且构造函数的参数通过byType进行装配。
- autodetect：自动探测，如果有构造方法，通过 construct的方式自动装配，否则使用 byType的方式自动装配。

基于注解的方式： 使用@Autowired注解来自动装配指定的bean。 在使用@Autowired注解之前需要在Spring配置文件进行配置，<context:annotation-config />。 在启动spring IoC时，容器自动装载了一个AutowiredAnnotationBeanPostProcessor后置处理器，当容器扫描到@Autowied、@Resource或@Inject时，就会在IoC容器自动查找需要的bean，并装配给该对象的属性。在使用@Autowired时，首先在容器中查询对应类型的bean：

> 如果查询结果刚好为一个，就将该bean装配给@Autowired指定的数据； 如果查询的结果不止一个，那么@Autowired会根据名称来查找； 如果上述查找的结果为空，那么会抛出异常。解决方法时，使用required=false。

@Autowired可用于：构造函数、成员变量、Setter方法 @Autowired和@Resource之间的区别 1、@Autowired默认是按照类型装配注入的，默认情况下它要求依赖对象必须存在（可以设置它required属性为false）。 2、@Resource默认是按照名称来装配注入的，只有当找不到与名称匹配的bean才会按照类型来装配注入。

Spring 框架中都用到了哪些设计模式

- 工厂模式：BeanFactory就是简单工厂模式的体现，用来创建对象的实例；
- 单例模式：Bean默认为单例模式。
- 代理模式：Spring的AOP功能用到了JDK的动态代理和CGLIB字节码生成技术；
- 模板方法：用来解决代码重复的问题。比如. RestTemplate, JmsTemplate, JpaTemplate。
- 观察者模式：定义对象键一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都会得到通知被制动更新，如Spring中listener的实现--ApplicationListener。

Spring事务的实现方式和实现原理

Spring事务的本质其实就是数据库对事务的支持，没有数据库的事务支持，spring是无法提供事务功能的。真正的数据库层的事务提交和回滚是通过binlog或者redo log实现的。

Spring事务的种类：

spring支持编程式事务管理和声明式事务管理两种方式：

- 编程式事务管理使用TransactionTemplate。
- 声明式事务管理建立在AOP之上的。其本质是通过AOP功能，对方法前后进行拦截，将事务处理的功能编织到拦截的方法中，也就是在目标方法开始之前加入一个事务，在执行完目标方法之后根据执行情况提交或者回滚事务。
  声明式事务最大的优点就是不需要在业务逻辑代码中掺杂事务管理的代码，只需在配置文件中做相关的事务规则声明或通过@Transactional注解的方式，便可以将事务规则应用到业务逻辑中。
  声明式事务管理要优于编程式事务管理，这正是spring倡导的非侵入式的开发方式，使业务代码不受污染，只要加上注解就可以获得完全的事务支持。唯一不足地方是，最细粒度只能作用到方法级别，无法做到像编程式事务那样可以作用到代码块级别。

spring的事务传播行为：

spring事务的传播行为说的是，当多个事务同时存在的时候，spring如何处理这些事务的行为。

- PROPAGATION_REQUIRED：如果当前没有事务，就创建一个新事务，如果当前存在事务，就加入该事务，该设置是最常用的设置。
- PROPAGATION_SUPPORTS：支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就以非事务执行。‘
- PROPAGATION_MANDATORY：支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就抛出异常。
- PROPAGATION_REQUIRES_NEW：创建新事务，无论当前存不存在事务，都创建新事务。
- PROPAGATION_NOT_SUPPORTED：以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。
- PROPAGATION_NEVER：以非事务方式执行，如果当前存在事务，则抛出异常。
- PROPAGATION_NESTED：如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则按REQUIRED属性执行。

Spring中的隔离级别：

- ISOLATION_DEFAULT：这是个 PlatfromTransactionManager 默认的隔离级别，使用数据库默认的事务隔离级别。
- ISOLATION_READ_UNCOMMITTED：读未提交，允许另外一个事务可以看到这个事务未提交的数据。
- ISOLATION_READ_COMMITTED：读已提交，保证一个事务修改的数据提交后才能被另一事务读取，而且能看到该事务对已有记录的更新。
- ISOLATION_REPEATABLE_READ：可重复读，保证一个事务修改的数据提交后才能被另一事务读取，但是不能看到该事务对已有记录的更新。
- ISOLATION_SERIALIZABLE：一个事务在执行的过程中完全看不到其他事务对数据库所做的更新。

Spring通知有哪些类型

1、前置通知Before advice：在某连接点join point之前执行的通知，但这个通知不能阻止连接点前的执行（除非它抛出一个异常）。 2、返回后通知After returning advice：在某连接点join point正常完成后执行的通知：例如一个方法没有抛出任何异常，正常返回。 3、抛出异常后通知After throwing advice：在方法抛出异常退出时执行的通知。 4、最终通知After (finally) advice ）：当某连接点退出的时候执行的通知（不论是正常返回还是异常退出）。 5、环绕通知Around Advice：包围一个连接点join point的通知，如方法调用。这是最强大的一种通知类型。 环绕通知可以在方法调用前后完成自定义的行为。它也会选择是否继续执行连接点或直接返回它们自己的返回值或抛出异常来结束执行。 环绕通知是最常用的一种通知类型。大部分基于拦截的AOP框架，例如Nanning和JBoss4，都只提供环绕通知。

同一个aspect，不同advice的执行顺序：

没有异常情况下的执行顺序：

> around before advice
>
> before advice
>
> target method 执行
>
> around after advice
>
> after advice
>
> afterReturning

有异常情况下的执行顺序：

> around before advice
>
> before advice
>
> target method 执行
>
> around after advice
>
> after advice
>
> afterThrowing:异常发生
>
> java.lang.RuntimeException: 异常发生

spring中的事务配置

1、在主类中声明打开事务支持

```
@EnableTransactionManagement
public class Test001Application {
```

2、在业务类和对应的方法上添加事务特性配置

```
@Service
@Transactional(readOnly = true, propagation = Propagation.SUPPORTS)
public class UserServImpl implements IUserServ {
    @Autowired
    private UserMapper userMapper;
    @Transactional(readOnly = false,propagation = Propagation.REQUIRED)
    public boolean create(User user) {
        调用userMapper执行插入操作
    }
    @Override
    public boolean login(User user) {
        调用userMapper执行对应的查询操作，这里事务特性将使用类上的声明
    }
}
```

MyBatis框架

什么是Mybatis

1、Mybatis是一个半自动化ORM对象关系映射框架，它内部封装了JDBC，开发时只需要关注SQL语句本身，不需要花费精力去处理加载驱动、创建连接、创建statement等繁杂的过程。程序员直接编写原生态sql，可以严格控制sql执行性能，灵活度高

2、MyBatis 可以使用 XML 或注解来配置和映射原生信息，将 POJO映射成数据库中的记录，避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集

3、通过xml 文件或注解的方式将要执行的各种 statement 配置起来，并通过java对象和 statement中sql的动态参数进行映射生成最终执行的sql语句，最后由mybatis框架执行sql并将结果映射为java对象并返回。（从执行sql到返回result的过程）。

Mybaits的优点：

1、基于SQL语句编程，相当灵活，不会对应用程序或者数据库的现有设计造成任何影响，SQL写在XML里，解除sql与程序代码的耦合，便于统一管理；提供XML标签，支持编写动态SQL语句，并可重用。

2、与JDBC相比，减少了50%以上的代码量，消除了JDBC大量冗余的代码，不需要手动开关连接；

3、很好的与各种数据库兼容（因为MyBatis使用JDBC来连接数据库，所以只要JDBC支持的数据库MyBatis都支持）。

4、能够与Spring很好的集成；

5、提供映射标签，支持对象与数据库的ORM字段关系映射；提供对象关系映射标签，支持对象关系组件维护。

MyBatis框架的缺点：

1、SQL语句的编写工作量较大，尤其当字段多、关联表多时，对开发人员编写SQL语句的功底有一定要求。

2、SQL语句依赖于数据库，导致数据库移植性差，不能随意更换数据库。

MyBatis框架适用场合：

1、MyBatis专注于SQL本身，是一个足够灵活的DAO层解决方案。

2、对性能的要求很高，或者需求变化较多的项目，如互联网项目，MyBatis将是不错的选择。

MyBatis与Hibernate有哪些不同？

1、Mybatis和hibernate不同，它不完全是一个ORM框架，因为MyBatis需要程序员自己编写Sql语句。

2、Mybatis直接编写原生态sql，可以严格控制sql执行性能，灵活度高，非常适合对关系数据模型要求不高的软件开发，因为这类软件需求变化频繁，一但需求变化要求迅速输出成果。但是灵活的前提是mybatis无法做到数据库无关性，如果需要实现支持多种数据库的软件，则需要自定义多套sql映射文件，工作量大。

3、Hibernate对象/关系映射能力强，数据库无关性好，对于关系模型要求高的软件，如果用hibernate开发可以节省很多代码，提高效率。

`#{}`和${}的区别是什么？

- `#{}`是预编译处理，`${}`是字符串替换。
- Mybatis在处理#{}时，会将sql中的#{}替换为?号，调用PreparedStatement的set方法来赋值；
- Mybatis在处理时，就是把{}替换成变量的值。
- 使用#{}可以有效的防止SQL注入，提高系统安全性。

通常一个Xml映射文件，都会写一个Dao接口与之对应，请问，这个Dao接口的工作原理是什么？Dao接口里的方法，参数不同时，方法能重载吗？

- Dao接口即Mapper接口。接口的全限名，就是映射文件中的namespace的值；接口的方法名，就是映射文件中Mapper的Statement的id值；接口方法内的参数，就是传递给sql的参数。
- Mapper接口是没有实现类的，当调用接口方法时，接口全限名+方法名拼接字符串作为key值，可唯一定位一个MapperStatement。在Mybatis中，每一个`<select>、<insert>、<update>、<delete>`标签，都会被解析

为一个MapperStatement对象。

> 举例：com.mybatis3.mappers.StudentDao.findStudentById，可以唯一找到namespace为 com.mybatis3.mappers.StudentDao下面 id 为 findStudentById 的 MapperStatement。

- Mapper接口里的方法，是不能重载的，因为是使用【全限名+方法名】的保存和寻找策略。Mapper 接口的工作原理是JDK动态代理，Mybatis运行时会使用JDK动态代理为Mapper接口生成代理对象proxy，代理对象会拦截接口方法，转而执行MapperStatement所代表的sql，然后将sql执行结果返回。

Mybatis是如何进行分页的？分页插件的原理是什么？

Mybatis使用RowBounds对象进行分页，它是针对ResultSet结果集执行的内存分页，而非物理分页。可以在sql内直接书写带有物理分页的参数来完成物理分页功能，也可以使用分页插件pagehelper来完成物理分页。

分页插件的基本原理是使用Mybatis提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的sql，然后重写sql，根据dialect方言，添加对应的物理分页语句和物理分页参数。

为什么说Mybatis是半自动ORM映射工具？它与全自动的区别在哪里？

Hibernate属于全自动ORM映射工具，使用Hibernate查询关联对象或者关联集合对象时，可以根据对象关系模型直接获取，所以它是全自动的。而Mybatis在查询关联对象或关联集合对象时，需要手动编写sql来完成，所以，称之为半自动ORM映射工具

MyBatis实现一对多有几种方式，怎么操作的？

有联合查询和嵌套查询。

联合查询是几个表联合查询,只查询一次,通过在resultMap里面的collection节点配置一对多的类就可以完成；

嵌套查询是先查一个表,根据这个表里面的 结果的外键id,去再另外一个表里面查询数据,也是通过配置collection,但另外一个表的查询通过select节点配置

Mybatis是否支持延迟加载？如果支持，它的实现原理是什么？

Mybatis仅支持association关联对象和collection关联集合对象的延迟加载，association指的就是一对一，collection指的就是一对多查询。在Mybatis配置文件中，可以配置是否启用延迟加载。例如lazyLoadingEnabled = true|false。

它的原理是，使用CGLIB创建目标对象的代理对象，当调用目标方法时，进入拦截器方法，比如调用a.getB().getName()，拦截器invoke()方法发现a.getB()是null值，那么就会单独发送事先保存好的查询关联B对象的sql，把B查询上来，然后调用a.setB(b)，于是a的对象b属性就有值了，接着完成a.getB().getName()方法的调用。这就是延迟加载的基本原理。

当然了，不光是Mybatis，几乎所有的包括Hibernate，支持延迟加载的原理都是一样的

Mybatis的一级、二级缓存:

- 一级缓存: 基于 PerpetualCache 的 HashMap 本地缓存，其存储作用域为 Session，当 Session flush 或 close 之后，该 Session 中的所有 Cache 就将清空，默认打开一级缓存
- 二级缓存与一级缓存其机制相同，默认也是采用 PerpetualCache，HashMap 存储，不同在于其存储作用域为 Mapper(Namespace)，并且可自定义存储源，如 Ehcache。默认不打开二级缓存，要开启二级缓存，使用二级缓存属性类需要实现Serializable序列化接口(可用来保存对象的状态),可在它的映射文件中配置 `<cache/>`

- 对于缓存数据更新机制，当某一个作用域(一级缓存 Session/二级缓存Namespaces)的进行了C/U/D 操作后，默认该作用域下所有 select 中的缓存将被 clear 掉并重新更新，如果开启了二级缓存，则只根据配置判断是否刷新

什么是MyBatis的接口绑定？有哪些实现方式？

接口绑定就是在MyBatis中任意定义接口,然后把接口里面的方法和SQL语句绑定, 我们直接调用接口方法就可以,这样比起原来了SqlSession提供的方法我们可以有更加灵活的选择和设置。

接口绑定有两种实现方式,一种是通过注解绑定，就是在接口的方法上面加上 @Select、@Update等注解，里面包含Sql语句来绑定；另外一种就是通过xml里面写SQL来绑定, 在这种情况下,要指定xml映射文件里面的namespace必须为接口的全路径名。当Sql语句比较简单时候,用注解绑定, 当SQL语句比较复杂时候,用xml绑定,一般用xml绑定的比较多

Mapper编写有哪几种方式

第一种：接口实现类继承SqlSessionDaoSupport：使用此种方法需要编写mapper接口，mapper接口实现类、mapper.xml文件。（1）在sqlMapConfig.xml中配置mapper.xml的位置

```
<mappers>
    <mapper resource="mapper.xml文件的地址" />
    <mapper resource="mapper.xml文件的地址" />
</mappers>
```

（2）定义mapper接口（3）实现类集成SqlSessionDaoSupport mapper方法中可以this.getSqlSession()进行数据增删改查。（4）spring 配置

```
<bean id=" " class="mapper接口的实现">
    <property name="sqlSessionFactory" ref="sqlSessionFactory"></property>
</bean>
```

第二种：使用org.mybatis.spring.mapper.MapperFactoryBean：（1）在sqlMapConfig.xml中配置mapper.xml的位置，如果mapper.xml和mappre接口的名称相同且在同一个目录，这里可以不用配置

```
<mappers>
    <mapper resource="mapper.xml文件的地址" />
    <mapper resource="mapper.xml文件的地址" />
</mappers>
```

（2）定义mapper接口：①mapper.xml中的namespace为mapper接口的地址 ②mapper接口中的方法名和mapper.xml中的定义的statement的id保持一致 ③Spring中定义

```
<bean id="" class="org.mybatis.spring.mapper.MapperFactoryBean">
    <property name="mapperInterface"   value="mapper接口地址" />
    <property name="sqlSessionFactory" ref="sqlSessionFactory" />
</bean>
```

第三种：使用mapper扫描器：（1）mapper.xml文件编写： mapper.xml中的namespace为mapper接口的地址； mapper接口中的方法名和mapper.xml中的定义的statement的id保持一致； 如果将mapper.xml和mapper接口的名称保持一致则不用在sqlMapConfig.xml中进行配置。 （2）定义mapper接口： 注意mapper.xml的文件名和mapper的接口名称保持一致，且放在同一个目录（3）配置mapper扫描器：

```
<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">
    <property name="basePackage" value="mapper接口包地址"></property>
    <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>
</bean>
```

（4）使用扫描器后从spring容器中获取mapper的实现对象。

简述Mybatis的插件运行原理，以及如何编写一个插件。

Mybatis仅可以编写针对ParameterHandler、ResultSetHandler、StatementHandler、Executor这4种接口的插件，Mybatis使用JDK的动态代理，为需要拦截的接口生成代理对象以实现接口方法拦截功能，每当执行这4种接口对象的方法时，就会进入拦截方法，具体就是InvocationHandler的invoke()方法，当然，只会拦截那些你指定需要拦截的方法。 编写插件：实现Mybatis的Interceptor接口并复写intercept()方法，然后在给插件编写注解，指定要拦截哪一个接口的哪些方法即可，记住，别忘了在配置文件中配置你编写的插件

具体开发配置

1、可以使用反向映射插件执行反向工程以生成xml映射元文件、Mapper接口、实体类

2、在主类上添加自动扫描注册Mapper接口

```
@MapperScan(basePackages = "com.yan.dao", markerInterface = SqlMapper.class)
public class Test001Application {
```

3、在应用配置文件application.yml中添加配置注册xml映射元文件

```
mybatis:
  mapper-locations:
  - classpath:com/yan/mapper/*.xml
```

4、在应用中配置打开输出mybatis所执行的sql语句，用于开发中查找错误

```
logging:
  level:
    com.yan: debug
```

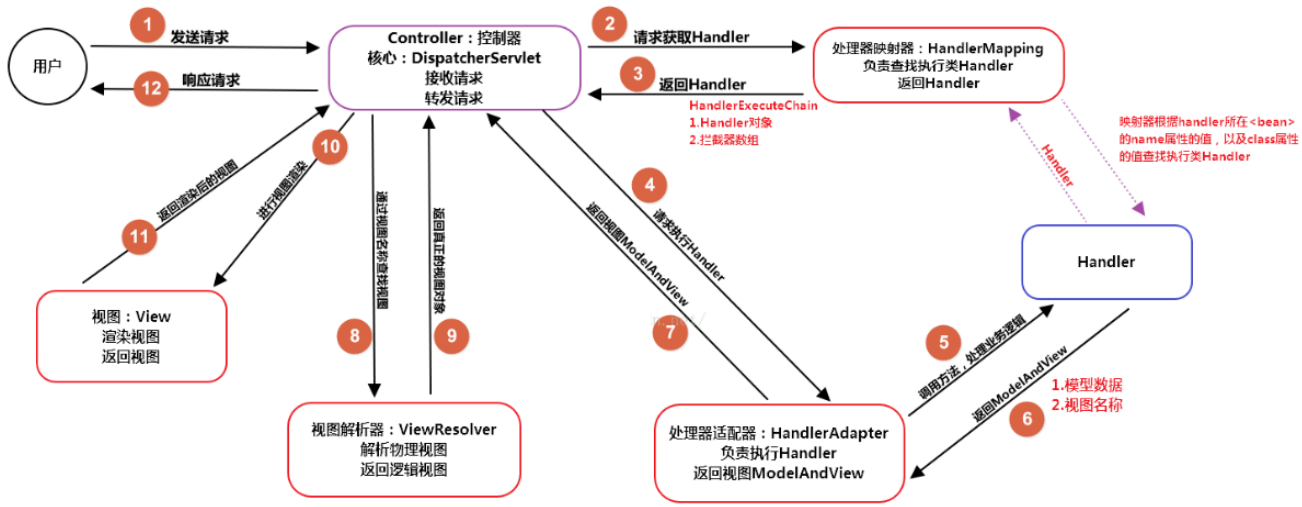
SpringMVC框架

什么是Spring MVC ？简单介绍下你对springMVC的理解?

Spring MVC是一个基于Java的实现了MVC设计模式的请求驱动类型的轻量级Web框架，通过把Model，View，Controller分离，将web层进行职责解耦，把复杂的web应用分成逻辑清晰的几部分，简化开发，减少出错，方便组内开发人员之间的配合。

SpringMVC的流程？

1）用户发送请求至前端控制器DispatcherServlet； 2） DispatcherServlet收到请求后，调用HandlerMapping处理器映射器，请求获取Handle； 3）处理器映射器根据请求url找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet； 4）DispatcherServlet 调用 HandlerAdapter处理器适配器； 5）HandlerAdapter 经过适配调用 具体处理器(Handler，也叫后端控制器)； 6）Handler执行完成返回ModelAndView； 7）HandlerAdapter将Handler执行结果ModelAndView返回给DispatcherServlet； 8）DispatcherServlet将ModelAndView传给ViewResolver视图解析器进行解析； 9）ViewResolver解析后返回具体View； 10）DispatcherServlet对View进行渲染视图（即将模型数据填充至视图中） 11）DispatcherServlet响应用户。



Springmvc的优点

1）可以支持各种视图技术,而不仅仅局限于JSP； 2）与Spring框架集成（如IoC容器、AOP等）； 3）清晰的角色分配：前端控制器dispatcherServlet , 请求到处理器映射handlerMapping, 处理器适配器HandlerAdapter, 视图解析器ViewResolver。 4） 支持各种请求资源的映射策略。

Spring MVC的主要组件？

1）前端控制器 DispatcherServlet，不需要程序员开发。作用：接收请求、响应结果，相当于转发器，有了DispatcherServlet 就减少了其它组件之间的耦合度。 2）处理器映射器HandlerMapping，不需要程序员开发。作用：根据请求的URL来查找Handler 3）处理器适配器HandlerAdapter。注意：在编写Handler的时候要按照HandlerAdapter要求的规则去编写，这样适配器HandlerAdapter才可以正确的去执行Handler。 4）处理器Handler，需要程序员开发。 5）视图解析器 ViewResolver，不需要程序员开发。作用：进行视图的解析，根据视图逻辑名解析成真正的视图view 6）视图View，需要程序员开发jsp。View是一个接口， 它的实现类支持不同的视图类型（jsp，freemarker，pdf等等）

springMVC和struts2的区别有哪些?

1）springmvc的入口是一个servlet即前端控制器DispatchServlet，而struts2入口是一个filter过虑器StrutsPrepareAndExecuteFilter。 2）springmvc是基于方法开发(一个url对应一个方法)，请求参数传递到方法的形参，可以设计为单例或多例(建议单例)，struts2是基于类开发，传递参数是通过类的属性，只能设计为多例。3）Struts采用值栈存储请求和响应的数据，通过OGNL存取数据，springmvc通过参数解析器是将request请求内容解析，并给方法形参赋值，将数据和视图封装成ModelAndView对象，最后又将ModelAndView中的模型数据通过reques域传输到页面。Jsp视图解析器默认使用jstl。

SpringMVC怎么样设定重定向和转发的？

1）转发：在返回值前面加"forward:"，譬如`"forward:user.do?name=method4"`

2）重定向：在返回值前面加"redirect:"，譬如`"redirect:http://www.baidu.com"`

SpringMvc怎么和AJAX相互调用的？

通过Jackson框架就可以把Java里面的对象直接转化成Js可以识别的Json对象。具体步骤如下 ：

> 1）加入Jackson.jar 2）在配置文件中配置json的映射 3）在接受Ajax方法里面可以直接返回Object,List等,但方法前面要加上@ResponseBody注解。

Spring MVC的异常处理 ？

可以将异常抛给Spring框架，由Spring框架来处理；我们只需要配置简单的异常处理器，在异常处理器中添视图页面即可。

SpringMvc的控制器是不是单例模式,如果是,有什么问题,怎么解决？

是单例模式,所以在多线程访问的时候有线程安全问题,不要用同步,会影响性能的,解决方案是在控制器里面不能写字段。

SpringMVC常用的注解有哪些？

@RequestMapping：用于处理请求 url 映射的注解，可用于类或方法上。用于类上，则表示类中的所有响应请求的方法都是以该地址作为父路径。 @RequestBody：注解实现接收http请求的json数据，将json转换为java对象。@ResponseBody：注解实现将conreoller方法返回对象转化为json对象响应给客户。

SpringMvc里面拦截器是怎么写的：

有两种写法,一种是实现HandlerInterceptor接口，另外一种是继承适配器类，接着在接口方法当中，实现处理逻辑；然后在SpringMvc的配置文件中配置拦截器即可：

```
<!-- 配置SpringMvc的拦截器 -->
<mvc:interceptors>
    <!-- 配置一个拦截器的Bean就可以了 默认是对所有请求都拦截 -->
    <bean id="myInterceptor" class="com.yan.action.MyHandlerInterceptor"></bean>
    <!-- 只针对部分请求拦截 -->
<mvc:interceptor>
    <mvc:mapping path="/modelMap.do" />
    <bean class="com.yan.action.MyHandlerInterceptorAdapter" />
</mvc:interceptor>
</mvc:interceptors>
```

注解原理：

注解本质是一个继承了Annotation的特殊接口，其具体实现类是Java运行时生成的动态代理类。我们通过反射获取注解时，返回的是Java运行时生成的动态代理对象。通过代理对象调用自定义注解的方法，会最终调用AnnotationInvocationHandler的invoke方法。该方法会从memberValues这个Map中索引出对应的值。而memberValues的来源是Java常量池。

微信登录

OAuth开发授权，允许第三方应用访问资源

需要多次访问微信的服务器，获取用户的openid，然后将openid存储在数据库中，通过openid查询数据库，如果数据库中有对应的openid则登录成功

作业

要求完成一个单表用户信息的CRUD操作，提交邮箱为19903152@qq.com