1. **J2EE**

J2EE是一套全然不同于传统应用开发的技术架构，包含许多组件，主要可简化且规范应用系统的开发与部署，进而提高可移植性、安全与再用价值。

J2EE核心是一组技术规范与指南，其中所包含的各类组件、服务架构及技术层次，均有共同的标准及规格，让各种依循J2EE架构的不同平台之间，存在良好的兼容性，解决过去企业后端使用的信息产品彼此之间无法兼容，企业内部或外部难以互通的窘境。

J2EE组件和“标准的” Java类的不同点在于：它被装配在一个J2EE应用中，具有固定的格式并遵守J2EE规范，由J2EE服务器对其进行管理。J2EE规范是这样定义J2EE组件的：客户端应用程序和applet是运行在客户端的组件；Java Servlet和Java Server Pages (JSP) 是运行在服务器端的Web组件；Enterprise Java Bean (EJB )组件是运行在服务器端的业务组件。

1. **J2EE三层结构**

典型的J2EE三层结构，分为表现层、中间层（控制层、业务逻辑层）和数据服务层。将业务规则、数据访问及合法性校验等工作放在中间层处理。客户端不直接与数据库交互，而是通过组件与中间层建立连接，再由中间层与数据库交互。表现层是传统的JSP技术。中间层采用的是流行的Spring+Hibernate，为了将控制与业务逻辑分离，又细分为以下几种：

Web层，就是控制层，框架采用Struts。

Service层（就是业务逻辑层），负责实现业务逻辑。业务逻辑层以DAO层为基础，通过对DAO组件包装，完成系统所要求的业务逻辑。

DAO层，负责与持久化对象交互。该层封装了数据的增、删、查、改的操作。

PO层，持久化对象。通过实体关系映射工具将关系型数据库的数据映射成对象，很方便地实现以面向对象方式操作数据库，采用Hibernate作为ORM框架。

Spring的作用贯穿了整个中间层，将Web层、Service层、DAO层及PO无缝整合，其数据服务层用来存放数据。

1. **Spring**
2. Spring介绍

Spring是一个开源框架，它由Rod Johnson创建。它是为了解决企业应用开发的复杂性而创建的。Spring使用基本的JavaBean来完成以前只可能由EJB完成的事情。然而，Spring的用途不仅限于服务器端的开发。从简单性、可测试性和松耦合的角度而言，任何Java应用都可以从Spring中受益。Spring是轻量级的Java EE应用程序框架。

Spring的核心是个轻量级容器（container），实现了IoC（Inversion of Control）模式的容器，Spring的目标是实现一个全方位的整合框架，在Spring框架下实现多个子框架的组合，这些子框架之间彼此可以独立，也可以使用其它的框架方案加以替代，Spring希望提供one-stop shop的框架整合方案Spring不会特別去提出一些子框架来与现有的OpenSource框架竞争，除非它觉得所提出的框架够新够好，例如Spring有自己的MVC框架方案，因为它觉得现有的MVC方案有很多可以改进的地方，但它不强迫您使用它提供的方案，您可以选用您所希望的框架来取代其子框架，例如您仍可以在Spring中整合您的Struts框架。

Spring的核心概念是IoC，IoC的抽象概念是依赖关系的转移，像是高层模组不应该依赖低层模组，而是模组都必须依赖于抽象是IoC的一种表现，实现必须依赖抽象，而不是抽象依赖实现也是IoC的一种表现，应用程序不应依赖于容器，而是容器服务于应用程序也是IoC的一种表现。struts负责控制Service（业务逻辑处理类），从而控制了Service的生命周期，这样层与层之间的依赖很强，属于耦合。这时，使用spring框架就起到了控制Action对象（Strus中的）和Service类的作用，两者之间的关系就松散了，Spring的Ioc机制（控制反转和依赖注入）正是用在此处。

1. Spring好处
2. Spring的Ioc（控制反转和依赖注入）

控制反转：就是由容器控制程序之间的（依赖）关系，而非传统实现中，由程序代码直接操控。

依赖注入：组件之间的依赖关系由容器在运行期决定 ，由容器动态的将某种依赖关系注入到组件之中 。 从上面我们不难看出：从头到尾Action仅仅是充当了Service的控制工具，这些具体的业务方法是怎样实现的，他根本就不会管，也不会问，他只要知道这些业务实现类所提供的方法接口就可以了。而在以往单独使用Struts框架的时候，所有的业务方法类的生命周期，甚至是一些业务流程都是由Action来控制的。层与层之间耦合性太紧密了，既降低了数据访问的效率又使业务逻辑看起来很复杂，代码量也很多。，Spring容器控制所有Action对象和业务逻辑类的生命周期，由于上层不再控制下层的生命周期，层与层之间实现了完全脱耦，使程序运行起来效率更高，维护起来也方便。

1. 使用Spring的第二个好处（AOP应用）

事务的处理：

在以往的JDBCTemplate中事务提交成功，异常处理都是通过Try/Catch 来完成，而在Spring中。Spring容器集成了TransactionTemplate，她封装了所有对事务处理的功能，包括异常时事务回滚，操作成功时数据提交等复杂业务功能。这都是由Spring容器来管理，大大减少了程序员的代码量，也对事务有了很好的管理控制。Hibernate中也有对事务的管理，hibernate中事务管理是通过SessionFactory创建和维护Session来完成。而Spring对SessionFactory配置也进行了整合，不需要在通过hibernate.cfg.xml来对SessionaFactory进行设定。这样的话就可以很好的利用Sping对事务管理强大功能。避免了每次对数据操作都要现获得Session实例来启动事务/提交/回滚事务还有繁琐的Try/Catch操作。这些也就是Spring中的AOP（面向切面编程）机制很好的应用。一方面使开发业务逻辑更清晰、专业分工更加容易进行。另一方面就是应用Spirng AOP隔离降低了程序的耦合性使我们可以在不同的应用中将各个切面结合起来使用大大提高了代码重用度

1. 优点总结：
2. 降低了组件之间的耦合性 ，实现了软件各层之间的解耦
3. 可以使用容易提供的众多服务，如事务管理，消息服务等
4. 容器提供单例模式支持
5. 容器提供了AOP技术，利用它很容易实现如权限拦截，运行期监控等功能
6. 容器提供了众多的辅助类，能加快应用的开发
7. spring对于主流的应用框架提供了集成支持，如hibernate，JPA，Struts等
8. spring属于低侵入式设计，代码的污染极低
9. 独立于各种应用服务器
10. spring的DI机制降低了业务对象替换的复杂性
11. Spring的高度开放性，并不强制应用完全依赖于Spring，开发者可以自由选择spring的部分或全部。
12. **Struts2**
13. Struts2基本介绍

Struts2是一个基于Sun Java EE平台的MVC框架，主要是采用Servlet和JSP技术来实现的。Struts框架可分为以下四个主要部分，其中三个就和MVC模式紧密相关：

模型（Model），本质上来说在Struts中Model是一个Action类（这个会在后面详细讨论），开发者通过其实现商业逻辑，同时用户请求通过控制器（Controller）向Action的转发过程是基于由struts-config.xml文件描述的配置信息的。

视图（View），View是由与控制器Servlet配合工作的一整套JSP定制标签库构成，利用她们我们可以快速建立应用系统的界面。

控制器（Controller），本质上是一个Servlet，将客户端请求转发到相应Action类。

1. **Struts2优点**
2. 实现了MVC模式，层次结构清晰，使程序员只需关注业务逻辑的实现。
3. 丰富的标签库，大大提高了开发的效率。
4. Struts2提供丰富的拦截器实现。
5. 通过配置文件，就可以掌握整个系统各个部分之间的关系。
6. 异常处理机制，只需在配置文件中配置异常的映射，即可对异常做相应的 处理
7. Struts2的可扩展性高。Struts2的核心jar包中由一个struts-default.xml文件，在该文件中设置了一些默认的bean,resultType类型，默认拦截器栈等，所有这些默认设置，用户都可以利用配置文件更改，可以更改为自己开发的bean，resulttype等。因此用户开发了插件的话只要很简单的配置就可以很容易的和Struts2框架融合，这实现了框架对插件的可插拔的特性。
8. 面向切面编程的思想在Strut2中也有了很好的体现。最重要的体现就是拦截器的使用，拦截器就是一个一个的小功能单位，用户可以将这些拦截器合并成一个大的拦截器，这个合成的拦截器就像单独的拦截器一样，只要将它配置到一个、Action中就可以。
9. **Hibernate**
10. Hibernate介绍

是一个开放源代码的对象关系映射框架，它对JDBC进行了非常轻量级的对象封装，使得Java程序员可以随心所欲的使用对象编程思维来操纵数据库。 Hibernate可以应用在任何使用JDBC的场合，既可以在Java的客户端程序实用，也可以在Servlet/JSP的Web应用中使用，最具革命意义的是，Hibernate可以在应用EJB的J2EE架构中取代CMP，完成数据持久化的重任。Hibernate的核心接口一共有5个，分别为:Session、SessionFactory、Transaction、Query和Configuration。这5个核心接口在任何开发中都会用到。通过这些接口，不仅可以对持久化对象进行存取，还能够进行事务控制。下面对这五个核心接口分别加以介绍。 Query和Criteria接口负责执行各种数据库查询。它可以使用HQL语言或SQL语句两种表达方式。

Session接口:Session接口负责执行被持久化对象的CRUD操作(CRUD的任务是完成与数据库的交流，包含了很多常见的SQL语句。)。但需要注意的是Session对象是非线程安全的。同时，Hibernate的session不同于JSP应用中的HttpSession。这里当使用session这个术语时，其实指的是Hibernate中的session，而以后会将HttpSesion对象称为用户session。

SessionFactory接口:SessionFactroy接口负责初始化Hibernate。它充当数据存储源的代理，并负责创建Session对象。这里用到了工厂模式。需要注意的是SessionFactory并不是轻量级的，因为一般情况下，一个项目通常只需要一个SessionFactory就够，当需要操作多个数据库时，可以为每个数据库指定一个SessionFactory。

Configuration接口:Configuration接口负责配置并启动Hibernate，创建SessionFactory对象。在Hibernate的启动的过程中，Configuration类的实例首先定位映射文档位置、读取配置，然后创建SessionFactory对象。

Transaction接口:Transaction接口负责事务相关的操作。它是可选的，开发人员也可以设计编写自己的底层事务处理代码。

Query和Criteria接口:Query和Criteria接口负责执行各种数据库查询。它可以使用HQL语言或SQL语句两种表达方式。

1. 优点
2. 对象/关系数据库映射(Basic O/R Mapping)它使用时只需要操纵对象，使开发更对象化，抛弃了数据库中心的思想，完全的面向对象思想。
3. 透明持久化(Persistent)带有持久化状态的、具有业务功能的单线程对象，此对象生存期很短。这些对象可能是普通的JavaBeans/POJO，这个对象没有实现第三方框架或者接口，唯一特殊的是他们正与（仅仅一个）Session相关联。一旦这个Session被关闭，这些对象就会脱离持久化状态，这样就可被应用程序的任何层自由使用。（例如，用作跟表示层打交道的数据传输对象。）
4. 事务Transaction (org.Hibernate.Transaction)应用程序用来指定原子操作单元范围的对象，它是单线程的，生命周期很短。它通过抽象将应用从底层具体的JDBC、JTA以及CORBA事务隔离开。某些情况下，一个Session之内可能包含多个Transaction对象。尽管是否使用该对象是可选的，但无论是使用底层的API还是使用Transaction对象，事务边界的开启与关闭是必不可少的。
5. 它没有侵入性，即所谓的轻量级框架。
6. 移植性会很好。
7. 缓存机制。提供一级缓存和二级缓存。
8. 简洁的HQL编程。
9. Hibernate缺点：
10. Hibernate在批量数据处理的时候是有弱势。
11. 针对某一对象(单个对象)简单的查\改\删\增，不是批量修改、删除，适合用Hibernate；而对于批量修改、删除，不适合用Hibernate，这也是OR框架的弱点;要使用数据库的特定优化机制的时候，不适合用Hibernate。
12. **MVC**
13. MVC基本介绍

MVC模式有Model，View，Controller三部分构成。Model模型。主要用来负责业务逻辑的处理，数据的保持。Model是MVC模式的核心部分，它也是一个应用需要实现的最主要的部分：进行业务逻辑的处理。View视图。负责数据的输出，画面的表示。Controller控制器。负责接收从视图发送过来的数据，同时控制Model与View部分。它的主要任务是控制Model与View，所以被称为控制器。

1. 与J2EE关系

J2EE三层：表示层，业务逻辑层，持久化层；一般都是由框架实现。

MVC 模型视图控制器，主要说的是表示层。

1. MVC优点
2. 耦合性低

视图层和业务层分离，这样就允许更改视图层代码而不用重新编译模型和控制器代码，同样，一个应用的业务流程或者业务规则的改变只需要改动MVC的模型层即可。因为模型与控制器和视图相分离，所以很容易改变应用程序的数据层和业务规则。

1. 重用性高

随着技术的不断进步，需要用越来越多的方式来访问应用程序。MVC模式允许使用各种不同样式的视图来访问同一个服务器端的代码，因为多个视图能共享一个模型，它包括任何WEB（HTTP）浏览器或者无线浏览器（wap），比如，用户可以通过电脑也可通过手机来订购某样产品，虽然订购的方式不一样，但处理订购产品的方式是一样的。由于模型返回的数据没有进行格式化，所以同样的构件能被不同的界面使用。

MVC使开发和维护用户接口的技术含量降低。

1. 部署快

使用MVC模式使开发时间得到相当大的缩减，它使程序员（Java开发人员）集中精力于业务逻辑，界面程序员（HTML和JSP开发人员）集中精力于表现形式上。

1. 可维护性高

分离视图层和业务逻辑层也使得WEB应用更易于维护和修改。

1. 有利软件工程化管理

由于不同的层各司其职，每一层不同的应用具有某些相同的特征，有利于通过工程化、工具化管理程序代码。控制器也提供了一个好处，就是可以使用控制器来联接不同的模型和视图去完成用户的需求，这样控制器可以为构造应用程序提供强有力的手段。给定一些可重用的模型和视图，控制器可以根据用户的需求选择模型进行处理，然后选择视图将处理结果显示给用户。

1. 缺点
2. 没有明确的定义

完全理解MVC并不是很容易。使用MVC需要精心的计划，由于它的内部原理比较复杂，所以需要花费一些时间去思考。同时由于模型和视图要严格的分离，这样也给调试应用程序带来了一定的困难。每个构件在使用之前都需要经过彻底的测试。

1. 不适合小型，中等规模的应用程序

花费大量时间将MVC应用到规模并不是很大的应用程序通常会得不偿失。

1. 增加系统结构和实现的复杂性

对于简单的界面，严格遵循MVC，使模型、视图与控制器分离，会增加结构的复杂性，并可能产生过多的更新操作，降低运行效率。

1. 视图与控制器间的过于紧密的连接

视图与控制器是相互分离，但却是联系紧密的部件，视图没有控制器的存在，其应用是很有限的，反之亦然，这样就妨碍了他们的独立重用。

1. 视图对模型数据的低效率访问

依据模型操作接口的不同，视图可能需要多次调用才能获得足够的显示数据。对未变化数据的不必要的频繁访问，也将损害操作性能。

1. 一般高级的界面工具或构造器不支持模式

改造这些工具以适应MVC需要和建立分离的部件的代价是很高的，会造成MVC使用的困难