目录

[1. MVC 2](#_Toc458544359)

[2. Spring框架 3](#_Toc458544360)

[ 基本概念 3](#_Toc458544361)

[ 模块化组成 3](#_Toc458544362)

[ Spring框架的好处？ 4](#_Toc458544363)

[ Spring框架解决的问题 4](#_Toc458544364)

[ Spring框架的核心思想 4](#_Toc458544365)

[ 装配 4](#_Toc458544366)

[ 什么是Spring beans? 4](#_Toc458544367)

[ 解释Spring支持的几种bean的作用域。 5](#_Toc458544368)

[ 一个 Spring Bean 定义 包含什么？ 5](#_Toc458544369)

[ 依赖注入IOC是什么，依赖注入的作用是什么？ 5](#_Toc458544370)

[ 有哪些不同类型的IOC（依赖注入）方式？ 6](#_Toc458544371)

[ 哪种依赖注入方式你建议使用，构造器注入，还是 Setter方法注入？ 6](#_Toc458544372)

[ Spring的AOP是什么含义，AOP的作用是什么？ 6](#_Toc458544373)

[ AOP中的术语 7](#_Toc458544374)

[ Spring MVC的工作原理 7](#_Toc458544375)

[ 在Spring框架中如何更有效地使用JDBC? 8](#_Toc458544376)

[ Spring在数据库访问中是如何处理异常的？ 9](#_Toc458544377)

[ Spring支持的事务管理类型 9](#_Toc458544378)

[ Spring框架的事务管理有哪些优点？ 9](#_Toc458544379)

[ 两种事务管理类型的比较？ 9](#_Toc458544380)

[3. RMI 9](#_Toc458544381)

[4. Hibernate 10](#_Toc458544382)

[ 简介 10](#_Toc458544383)

[ 五个核心接口 10](#_Toc458544384)

[ 使用流程 10](#_Toc458544385)

[ 优点 10](#_Toc458544386)

[ 缺点 10](#_Toc458544387)

[ 使用Hibernate时如何提高性能？ 11](#_Toc458544388)

[ 二级缓存 11](#_Toc458544389)

[ 二级缓存的使用情况 11](#_Toc458544390)

[ JDBC与Hibern的区别 11](#_Toc458544391)

1. MVC

<1> MVC概念

MVC是Modle(模型)、 View(视图)、Controller(控制器)三个单词的首字母组合。是一种目前广泛流行的应用模型，其目的是实现web系统的职能分工。

(1)模型层

M：模型层主要实现系统中的业务逻辑，通常用JavaBeans或EJB来实现。

模型表示的是企业数据和业务逻辑，是应用程序的主体部分。业务流程的操作对其它层来说是黑盒子，模型接收视图的请求数据，并返回最终的结果。业务模型的设计可以说是MVC最主要的部分。

MVC把应用的模型按一定的规则抽象出来，抽象不能与具体隔得太远，以便于提高模型的重构和提高重用性。

业务模型还有一个很重要的模型，那就是数据模型。数据模型主要指实体对象的数据持续化，比如把一张订单保存到数据库，从数据库中获取订单，将这个模型单独列出，所有相关数据操作只限制在该模型中。

(2) 视图层

V：视图层主要用于与用户的交互，通常使用JSP来实现。视图是用户看到的并与之交互的界面。

MVC架构一个大的好处是它能够为Web应用处理很多不同的视图。在视图中其实没有真正的业务处理发生，不管这些数据是联机存储还是一个雇员列表，作为视图只是作为一种输出数据并允许用户操纵的方式。

视图的功能强大，主要表现在以下两个方面：

① 根据客户类型显示信息；

② 显示商业逻辑(模型)的结果，而不关心信息如何获得何时获得。

(3) 控制器

C：控制层则是模型与视图之间的桥梁，它可以把用户的请求分派并选择恰当的视图来显示处理结果，也可以解释用户的输入并将其映射为模型层能够执行的操作。

控制层接收用户的输入并调用模型和视图去完成用户的需求。所以当用户单击web页面中的超链接和发送HTML表单时，控制器(如servlet)本身不输出任何也不执行任何处理，只是接收请求并决定调用哪个模型去处理请求，然后确定哪个视图来显示模型处理所返回的数据。

<2> MVC处理流程

对于用户输入的请求，先被控制器接收，并决定由哪个模型来进行处理，然后模型通过业务逻辑层去处理用户的请求并返回数据，最后控制器用相应的视图格式化模型返回的数据，并通过页面展示给用户。

<3>MVC模型的优点

① 低耦合性：视图层和业务层分离，改变其中一个对另外一个影响不大，不改变的那个不需要重新编译；

② 较低的生命周期成本：MVC使开发和维护用户接口的技术难度降低；

③ 可重用性和可适应性：同样的模型层构建能够被不同的界面使用；

④ 部署快速：后台开发人员集中精力于业务逻辑上，界面开发人员集中精力于表现形式上；

⑤ 可维护性：业务层和视图层分离利于web应用维护；

⑥ 有利于软件工程化管理。

1. Spring框架

* 基本概念

Spring是一个开源框架，是为了解决企业级应用开发的复杂性而创建的。Spring中对象无需自己负责查找或创建与其关联的其他对象。相反，容器负责把需要相互协作的对象引用赋予各个对象。

Spring框架提供了对轻量级IoC的良好支持，同时也提供了对AOP技术非常好的封装。相比其他框架，Spring框架的设计更加模块化，框架内的每个模块都能完成特定的工作，而且各个模块也都可以独立运行，不会相互牵制。因此在使用Spring框架时，既可以使用整个框架，也可以使用其中的一部分。

* 模块化组成

Spring框架主要由Spring AOP、Spring ORM、Spring DAO、Spring Web、Spring Context、Spring Web MVC、Spring Core等几个模块组成。

|  |  |
| --- | --- |
| Spring AOP | 在AOP模块中，Spring对切面编程提供了丰富的支持。该模块是Spring应用系统开发切面的基础，借助于AOP，可将应用中的关注点(如事务和安全)从它们所应用的对象中解耦出来。 |
| Spring ORM | 提供了对现有ORM框架的支持，如Hibernate、JDO |
| Spring DAO | 提供了对数据库访问对象模式和JDBC的支持，DAO可以把实现逻辑与数据库访问代码相分离，从而降低了代码耦合度。通过对JDBC的抽象，简化了开发工作，同时简化了对异常的处理。 |
| Spring Web | 提供了serlvet监听器的context和web应用的上下文，同时还集成了一些现有的web框架如strusts。 |
| Spring Context | 扩展核心容器，提高了spring上下文环境，给开发人员提供了很多非常有用的服务。 |
| Spring Web MVC | 提供了一个构建web应用MVC框架 |
| Spring Core | Spring框架的核心容器，最主要的是BeanFactory组件，使用了工厂模式创建所需的对象，BeanFactory采用依赖注入的思想通过读取xml文件的信息来实例化对象。BeanFactory提供了组件声明周期的管理、组件的创建、装配和销毁等功能。 |

ORM：Object Relation Mapping 对象关系映射表

JDO：Java Data Object是Java对象持久化的新的规范，也是一个用于存取某种数据仓库中的对象的标准化API。

* Spring框架的好处？

① **轻量**：Spring 是轻量的，基本的版本大约2MB。

② **控制反转IOC**：Spring通过控制反转实现了松散耦合，对象给出它们的依赖，而不是创建或查找所依赖的对象。

③ **面向切面的编程(AOP)**：Spring支持面向切面的编程，把应用业务逻辑和系统服务分开。

④ **MVC框架**：Spring的MVC框架是个精心设计的框架，是Web框架的一个很好的替代品。

⑤ **事务管理**：Spring 提供一个持续的事务管理接口，可以扩展到上至本地事务下至全局事务（JTA）。

⑥ **异常处理**：Spring 提供方便的API把具体技术相关的异常（比如由JDBC，Hibernate or JDO抛出的）转化为一致的unchecked 异常

⑦ 这种方式最大的优点是各组件的耦合极为松散，并且无需我们自己实现Singleton模式。

* Spring框架解决的问题

① Spring 解决了的最核心的问题就是把对象之间的依赖关系转为用配置文件来管理，也就是Spring的依赖注入机制。这个注入机制是在Ioc容器中进行管理的。

② 设计模块化，各个模块可以独立运行。简化开发。

* Spring框架的核心思想

依赖注入Ioc和面向切面编程AOP。

Spring框架关注于通过依赖注入、面向切面编程和消除样板代码来简化企业级Java开发。

* 装配

装配指的是创建应用组件之间协作关系的行为。Spring中有多种装配Bean的行为，采用XML配置是最常见的装配方式。还有一种装配方式是Java注解的形式。

* 什么是Spring beans?

Spring beans 是那些形成Spring应用的主干的java对象。它们被Spring IOC容器初始化，装配和管理。这些beans通过容器中配置的元数据创建。比如，以XML文件中<bean/> 的形式定义。

Spring 框架定义的beans默认都是单例beans。在bean tag中有个属性”singleton”，如果它被赋为 TRUE，bean 就是单例，否则就是一个 prototype bean。默认是TRUE，所以所有在Spring框架中的beans 缺省都是单例，也可以将作用域设定为其他的类型。

* 解释Spring支持的几种bean的作用域。

当定义一个<bean> 在Spring里，我们还能给这个bean声明一个作用域。它可以通过bean 定义中的scope属性来定义。

Spring框架支持以下五种bean的作用域：

① **singleton** : bean在每个Spring ioc 容器中只有一个实例。

② **prototype**:一个bean的定义可以有多个实例。

③ **request**：每次http请求都会创建一个bean，该作用域仅在基于web的Spring ApplicationContext情形下有效。

④ **session**：在一个HTTP Session中，一个bean定义对应一个实例。该作用域仅在基于web的Spring ApplicationContext情形下有效。

⑤ **global-session**：在一个全局的HTTP Session中，一个bean定义对应一个实例。该作用域仅在基于web的Spring ApplicationContext情形下有效。

* 一个 Spring Bean 定义 包含什么？

一个Spring Bean 的定义包含容器必知的所有配置元数据，包括如何创建一个bean，它的生命周期详情及它的依赖。

* 依赖注入IOC是什么，依赖注入的作用是什么？

IOC(Inverse of Control,Ioc)，DI（Dependency Injection）叫依赖注入，是对IOC更简单的叙述。控制反转即把传统上在程序代码中直接操控对象的调用权转交给容器，通过容器来实现对象组件的装配和管理。即控制反转就是对组件对象控制权的转移，从程序代码转移到了外部容器，由容器来创建对象并管理对象之间的依赖关系。

DI是对IOC容器更准确的描述，即组件之间的依赖关系由容器在运行期间决定，更形象点说是由容器动态的将某种依赖关系注入到组件中。

作用：采用IoC机制能够提高系统的可扩展性，主要用于创建对象和管理对象。如果对象之间通过显示调用进行交互导致调用者与被调用者之间的关系太过紧密，任何一方的改变就会导致程序出现大的改动。

实现原理：

① 通过反射获取实例；

② 获取需要注入的接口实现类并将其赋值给该接口。

优点：

① 通过IoC容器，开发人员不需要关注对象如何被创建的，同时增加新类也非常方便，只需要修改配置文件即可实现对象的“热插拔”；

② IoC容器可以通过配置文件来确定需要注入的实例化对象，因此非常便于进行单元测试。

缺点：

① 对象是通过反射机制实例化出来的，因此对系统的性能有一定的影响；

② 创建对象的流程变得比较复杂。

其他补充：

Spring采用IoC的方式来实现把一个实例化的对象注入到用户自定义的对象中，具有较强的可扩展性。

通过依赖注入，对象的依赖关系将由负责协调系统中各个对象的第三方组件在创建对象时设定。对象无需自行创建或管理它们的依赖关系---依赖关系将被自动注入到需要它们的对象中去。

控制反转(Inverse of Control,Ioc)有时也被称为依赖注入，是一种降低对象之间耦合关系的设计思想。通过IoC的方式，使得上层程序不再依赖下层的接口，即通过采用一定的机制来选择不同的下层实现，完成控制反转，使得调用者控制被调用者。IoC通过注入一个实例化的对象来达到解耦和的目的。使用这种方法后，对象不会被显示地调用，而是根据需求通过IoC容器(Spring)来提供。

* 有哪些不同类型的IOC（依赖注入）方式？

① 构造器依赖注入：构造器依赖注入通过容器触发一个类的构造器来实现的，该类有一系列参数，每个参数代表一个对其他类的依赖。

② Setter方法注入：Setter方法注入是容器通过调用无参构造器或无参static工厂方法实例化bean之后，调用该bean的setter方法，即实现了基于setter的依赖注入。

* 哪种依赖注入方式你建议使用，构造器注入，还是 Setter方法注入？

两种依赖方式都可以使用，构造器注入和Setter方法注入。最好的解决方案是用构造器参数实现强制依赖，setter方法实现可选依赖。

* Spring的AOP是什么含义，AOP的作用是什么？

面向切面编程(Aspect Oriented Programming , AOP)是面向对象开发的一种补充，它允许开发人员在不改变原来模型的基础上动态地修改模型以满足新的需求。比如开发人员可以在不改变原来业务逻辑模型的基础上可以动态地增加日志、安全或异常处理功能。

通过面向切面编程，可以把之前分散在应用各处的行为放入可重用的模块中，显式地声明在何处如何应用该行为，这就有效地减少了代码的冗余，并让代码中的类更关注其自身的主要功能。

Spring的AOP适合于那些具有横切逻辑的应用：如性能监测，访问控制，事务管理、缓存、对象池管理以及日志记录。AOP将这些分散在各个业务逻辑中的代码通过横向切割的方式抽取到一个独立的模块中。AOP 实现的关键就在于 AOP 框架自动创建的 AOP 代理，AOP 代理则可分为静态代理和动态代理两大类，其中静态代理是指使用 AOP 框架提供的命令进行编译，从而在编译阶段就可生成 AOP 代理类，因此也称为编译时增强；而动态代理则在运行时借助于 JDK 动态代理、CGLIB 等在内存中“临时”生成 AOP 动态代理类，因此也被称为运行时增强。

代理对象的方法 = 增强处理 + 被代理对象的方法

优点：简单，易用

缺点：性能略低，仅适用于方法调用，必须在Spring容器。

作用：分布于应用中多处的功能被称为“横切关注点”，通常这些横切关注点从概念上是与应用的业务逻辑相分离的。将这些横切关注点与业务逻辑相分离正是面向切面编程(AOP)所要解决的。

Spring提供的AOP框架，让我们把切面插入方法执行的周围。

横切关注点可以被模块化为特殊的类，这些类被称为切面。

切面提供了取代继承和委托的另一种选择，而且在很多场景下更清晰简洁。在使用面向切面编程时，我们仍然在一个地方定义通用功能，但是我们可以通过声明的方式定义这个功能以何种方式在何处应用，而无需修改受影响的类。

它可以通过预编译方式和运行期动态代理实现在不修改源代码的情况下给程序动态统一添加功能的一种技术。

好处：① 每个关注点现在都只集中于一处，而不是分散到多处代码中；

② 服务模块更简洁，因为它们只包含主要关注点(或核心功能)的代码，而次要关注的代码都被移到切面中了。

* AOP中的术语

通知(Advice)：切面的工作被称为通知，通知定义了切面是什么以及何时使用；

连接点(Joinpoint)：连接点是在应用执行过程中能够插入切面的一个点，这个点可以是调用方法时、抛出异常时、甚至是修改一个字段时。切面代码可以利用这些点插入到应用的正常路程之中，并添加新的行为。

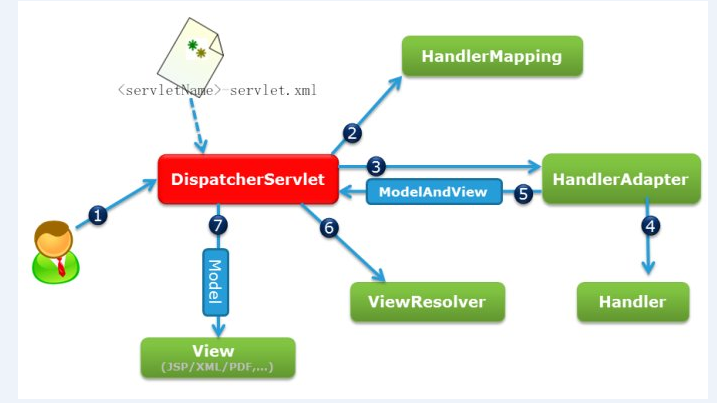
切点(PointCut)：如果通知定义了切面的“什么”和“何时”，那么切点就定义了“何处”。切点的定义会匹配通知所要织入的一个或多个连接点。一个切面并不需要通知应用的所有连接点，切点有助于缩小切面所通知连接点的范围。

切面(Aspect)：切面是通知和切点的结合，通知和切点共同定义了关于切面的全部内容，它是什么，在何时和何处完成其功能。

引入(Introduction)：引入允许我们向现有的类添加新方法或属性。

织入(Weaving)：织入是将切面应用到目标对象来创建新的代理对象的过程，切面在指定的连接点被织入到目标对象中。在目标对象的生命周期里有多个点可以进行织入。

* Spring MVC的工作原理

  
上面的是springMVC的工作原理图：

1、客户端发出一个http请求给web服务器，web服务器对http请求进行解析，如果匹配DispatcherServlet的请求映射路径（在web.xml中指定），web容器将请求转交给DispatcherServlet.

2、DipatcherServlet接收到这个请求之后将根据请求的信息（包括URL、Http方法、请求报文头和请求参数Cookie等）以及HandlerMapping的配置找到处理请求的处理器（Handler）。

3-4、DispatcherServlet根据HandlerMapping找到对应的Handler,将处理权交给Handler（Handler将具体的处理进行封装），再由具体的HandlerAdapter对Handler进行具体的调用。

5、Handler对数据处理完成以后将返回一个ModelAndView()对象给DispatcherServlet。

6、~~Handler返回的ModelAndView()只是一个逻辑视图并不是一个正式的视图~~，DispatcherSevlet通过ViewResolver将逻辑视图转化为真正的视图View。

7、Dispatcher通过model解析出ModelAndView()中的参数进行解析最终展现出完整的view并返回给客户端。

* 在Spring框架中如何更有效地使用JDBC?

使用SpringJDBC 框架，资源管理和错误处理的代价都会被减轻。所以开发者只需写statements 和 queries从数据库存取数据，JDBC也可以在Spring框架提供的模板类的帮助下更有效地被使用，这个模板叫JdbcTemplate

Spring的JDBC模板能够让开发人员不再去处理一些样板式代码(比如管理连接资源、处理异常等)，只需要处理关注实际的更新和查询即可。JDBC模板类提供了很多便利的方法解决诸如把数据库数据转变成基本数据类型或对象，执行写好的或可调用的数据库操作语句，提供自定义的数据错误处理。

通过使用JDBC抽象和DAO模块，保证数据库代码的简洁，并能避免数据库资源错误关闭导致的问题，它在各种不同的数据库的错误信息之上，提供了一个统一的异常访问层。它还利用Spring的AOP 模块给Spring应用中的对象提供事务管理服务。

* Spring在数据库访问中是如何处理异常的？

Spring认为触发异常的很多问题不能在catch代码块中修复，Spring使用了非检查型异常，而不是强制开发人员编写catch代码块。这把是否要捕获异常的权利留给了开发人员。

* Spring支持的事务管理类型

Spring支持两种类型的事务管理：

**编程式事务管理**：这意味你通过编程的方式管理事务，给你带来极大的灵活性，但是难维护。允许用户在代码中精确定义事务的边界，完全控制事务从哪里开始、在哪里提交以及在哪里结束。适用于那些完全想完全控制事务的边界，该事务类型为侵入式的，必须修改Spring的特定类来得到Spring的编码式事务的支持。

**声明式事务管理：**这意味着你可以将业务代码和事务管理分离，你只需用注解和XML配置来管理事务。通过使用Spring AOP框架来完成。声明式事务的方式是使用Spring的tx命名空间进行命名xml文件配置和@Transaction注解。

* Spring框架的事务管理有哪些优点？

① Spring将事务管理平台抽象为通用的API，从而让开发人员避免直接与特定的事务管理实现进行交互。

② Spring的声明式事务管理不仅能够在POJO上定义传播行为，还能声明隔离级别、只读优化以及特定异常的回滚规则。

* 两种事务管理类型的比较？

大多数Spring框架的用户选择声明式事务管理，因为它对应用代码的影响最小，因此更符合一个无侵入的轻量级容器的思想。声明式事务管理要优于编程式事务管理，虽然比编程式事务管理（这种方式允许你通过代码控制事务）少了一点灵活性。

1. RMI

RMI，远程方法调用（Remote Method Invocation）是Enterprise JavaBeans的支柱，是建立分布式Java应用程序的方便途径。RMI是非常容易使用的，但是它非常的强大。

RMI的基础是接口，RMI构架基于一个重要的原理：定义接口和接口的具体实现是分开的。

RMI主要用于远程调用服务，通过它可以像调用本地方法一样调用另外一台计算机上的程序，而不需要关心另外一台机子上所使用的计算机系统或所使用的语言，也就是说，只要有一定的规范，就可以实现在不同的计算机上进行函数调用。

远程调用其实是客户端应用和服务端之间的会话。在客户端，它所需要的一些功能并不在该应用的实现范围之内，所以应用向能提供这些功能的其他系统寻求帮助，而远程应用通过远程服务发布这些功能。

1. Hibernate

* 简介

Hibernate是一个开放源代码的对象关系映射(Object Relation Mapping , ORM,一种用来完成对象模型到关系模型的映射技术)框架。它不仅可以运行在J2EE容器中，也可以在J2EE容器外运行。对JDBC进行了非常轻量级的对象封装，所以可以在任何使用JDBC的地方都可以用Hibernate代替。

Hibernate实现了java代码与数据库中记录的映射关系，简化了程序开发人员访问数据库的流程，极大地提高了软件的开发效率。

* 五个核心接口

Hibernate主要提供了五个核心接口，分别为：Session、SessionFactory、Transcation、Query、Configuration。通过使用这些接口不仅可以完成对数据库的访问(例如查询、插入、更新和删除)，而且还可以实现对事务的控制。

* 使用流程

Hibernate的使用流程如下：

Step1: 应用程序通过Configuration类读取配置文件并创建SessionFactory对象；

Step2: 通过SessionFactory生成一个Session对象；

Step3: 通过Session对象的beginTrancation()方法创建一个事务，接着可以通过Session对象的get()，load()，save(), update()，delete()和saveOrUpdate()等方法实现数据的加载、保存、更新和删除等操作，也可以通过Session对象生成一个Query对象，然后利用Query对象执行查询操作，最后通过commit()或rollback()方法完成事物操作。

Step4: 在完成所有持久化操作与事务操作后关闭Session和SessionFactory对象。

* 优点

1. 提高开发效率；
2. 使得开发可以完全采用面向对象的思想，不需要关心数据库的关系模型；
3. 具有较好的可移植性，可以很容易地实现不同数据库之间的移植而不需要关心不同数据库SQL语句的差别；（？？？为什么不需要关心不同数据库的SQL语句的差别？？？？）
4. 支持透明持久化，Hibernate的API没有侵入性，当保存一个对象时，这个对象不需要继承Hibernate的任何类和实现接口。

* 缺点

1. 只适用于针对单一对象简单的增、删、改、查，而对于批量的修改/删除的场合，则不适用。

* 使用Hibernate时如何提高性能？

1. 延迟加载：当Hibernate从数据库中获取某一个对象数据、获取某个对象的集合属性值时或者获取某一个对象所关联的另一个对象时，并不会立即从数据库中加载到对象中，而是通过建立一个代理对象，把这个对象的属性都设为默认值，只有当这些数据在被使用时，才从数据库中加载对应的数据，采用这种办法有助于提高Hibernate的性能。
2. 缓存技术：Hibernate中提供了一级缓存与二级缓存，合理的利用缓存有助于提高系统的性能，为了避免不合理的利用缓存导致内存的过度消耗降低系统的性能，也可以通过合理配置缓存的参数来避免这个问题。
3. 优化查询语句：通过优化查询语句提高系统的性能。

* 二级缓存

一级缓存由Session管理，二级缓存由SessionFactory管理。在使用时一级缓存是必不可少的，二级缓存可有可无。

一级缓存使用的场合是：当使用session查询数据时，首先会在该session内部查找该对象是否存在，如果存在，则直接返回，否则，就到数据库中去查询，并将查询的结果缓存起来以便以后使用。一级缓存的缺点就是当使用session来表示一次会话时，它的生命周期较短，而且它是线程不安全的，不能被多个线程共享，因此，在实际使用时，对效率的提升并不是很明显。

鉴于以上的原因，就引入了二级缓存。

二级缓存是为Hibernate配置一种全局的缓存，以便实现多个线程与事务共享。在使用了二级缓存之后，当查询数据时，会首先在内部缓存中查找，如果不存在，接着在二级缓存中查找，最后在数据库中查找。与一级缓存相比，二级缓存是独立于Hibernate的软件部件，属于第三方产品，常见的有EhCache，OSCache和JbossCache等。在使用时可以根据需求通过配置二级缓存插件来实现二级缓存功能，Hibernate为了集成这些插件，提供了相应的接口来充当缓存插件与Hibernate之间的适配器。

* 二级缓存的适用情况

1. 数据量较小：如果数据量太大，缓存太多，会消耗大量的内存，造成内存资源短缺，影响系统性能；
2. 对数据的修改较少：如果进行大量的修改，就需要频繁地对缓存中的数据与数据库中的数据进行同步，这也会影响系统的性能；
3. 不会被大量的应用共享的数据：如果被大量的应用共享，多线程之间的同步也会影响性能；
4. 不是很重要的数据：如果查询的数据非常重要如财务数据，对数据的正确性要求非常高，最好不要使用二级缓存。

* JDBC与Hibernet的区别

Hibernate是JDBC的封装，采用配置文件的形式将数据库的连接参数写到XML文件中，至于对数据库的访问还是通过JDBC来访问的。

Hibernate的查询语句返回的是List<Object[]>类，而JDBC通过statement返回的查询结果是ResultSet并且有时候需要自己封装到List中。

Hibernate具有很好的维护性和扩展性。