# JAVA高级

## 1. 设计模式

### 1. (★)单例设计模式

答:

单例模式有以下特点：

　　1、单例类只能有一个实例。

　　2、单例类必须自己创建自己的唯一实例。

3、单例类必须给所有其他对象提供这一实例。 懒汉式 和 饿汉式

### 2. (★)介绍一下简单工厂模式

答:

简单工厂模式是属于创建型模式，又叫做静态工厂方法（Static Factory Method）模式，但不属于23种GOF设计模式之一。简单工厂模式是由一个工厂对象决定创建出哪一种产品类的实例。简单工厂模式是工厂模式家族中最简单实用的模式，可以理解为是不同工厂模式的一个特殊实现。

### 3. (★)简述一下代理模式(Proxy模式)

答:

Proxy是比较有用途的一种模式，而且变种较多，应用场合覆盖从小结构到整个系统的大结构，Proxy是代理的意思，我们也许有代理服务器等概念，代理概念可以解释为：在出发点到目的地之间有一道中间层，意为代理。

设计模式中定义：为其他对象提供一种代理以控制对这个对象的访问。

**为什么要使用代理模式**

授权机制 不同级别的用户对同一对象拥有不同的访问权利，如Jive论坛系统中，就使用Proxy进行授权机制控制，访问论坛有两种人：注册用户和游客(未注册用户)，Jive中就通过类似ForumProxy这样的代理来控制这两种用户对论坛的访问权限。

某个客户端不能直接操作到某个对象，但又必须和那个对象有所互动。

举例两个具体情况：

如果那个对象是一个是很大的图片，需要花费很长时间才能显示出来，那么当这个图片包含在文档中时，使用编辑器或浏览器打开这个文档，打开文档必须很迅速，不能等待大图片处理完成，这时需要做个图片Proxy来代替真正的图片。

如果那个对象在Internet的某个远端服务器上，直接操作这个对象因为网络速度原因可能比较慢，那我们可以先用Proxy来代替那个对象。

总之原则是，对于开销很大的对象，只有在使用它时才创建，这个原则可以为我们节省很多宝贵的Java内存。所以，有些人认为Java耗费资源内存，我以为这和程序编制思路也有一定的关系。

如何使用代理模式

以论坛系统为例，访问论坛系统的用户有多种类型：注册普通用户、论坛管理者、系统管理者、游客。注册普通用户才能发言，论坛管理者可以管理他被授权的论坛，系统管理者可以管理所有事务等，这些权限划分和管理是使用Proxy完成的。

### 4. (★★)常见的设计模式,任选模式一种模模式代码的实现

答:

单例,模板,工厂,装饰者,适配器,代理,观察者等设计模式

public class HungrySingle {

private static HungrySingle instance = new HungrySingle();

private HungrySingle() {

}

public static HungrySingle getInstance() {

return instance;

}

}

### 5. (★★)谈谈对 mvc 的认识

由模型(model),视图(view),控制器(controller)完成的应用程序由模型发出要实现的功能到控制器,控制器接收组织功能传递给视图;

MVC 是一个设计模式，它强制性的使应用程序的输入、处理和输出分开。使用 MVC 应用程序被分成三个核心部件：模型、视图、控制器。它们各自处理自己的任务。视图是用户看到并与之交互的界面。模型表示企业数据和业务规则。控制器接受用户的输入并调用模型和视图去完成用户的需求。

MVC 的优点：低耦合性、高重用性和可适用性、较低的生命周期成本、快速的部署、可维护性、可扩展性，有利于软件工程化管理

MVC 的缺点：没有明确的定义，完全理解 MVC 并不容易。不适合小型规模的应用程序。

### 6.(★★)简述一下你了解的设计模式。

（1） 工厂设计模式（Factory Method）：

常用的工厂模式是静态工厂，利用static方法，作为一种类似于常见的工具类utils等辅助效果，一般情况下工厂类不需要实例化

（2） 抽象工厂模式（Abstract Factory）：

一个基础接口定义了功能，每个实现接口的子类就是产品，然后定义一个工厂接口，实现了工厂接口的就是工厂，这时候，接口编程的优点就出现了，我们可以新增产品类（只需要实现产品接口），只需要同时新增一个工厂类，客户端就可以轻松调用新产品的代码。

抽象工厂的灵活性就体现在这里，无需改动原有的代码，毕竟对于客户端来说，静态工厂模式在不改动StaticFactory类的代码时无法新增产品，如果采用了抽象工厂模式，就可以轻松的新增拓展类。

（3） 单例模式（Singleton）：

在内部创建一个实例，构造器全部设置为private，所有方法均在该实例上改动，在创建上要注意类的实例化只能执行一次，可以采用许多种方法来实现，如Synchronized关键字，或者利用内部类等机制来实现。

（4） 装饰模式（Decorator）：

给一类对象增加新的功能，装饰方法与具体的内部逻辑无关

（5） 代理模式（Proxy）：

客户端通过代理类访问，代理类实现具体的实现细节，客户只需要使用代理类即可实现操作。这种模式可以对旧功能进行代理，用一个代理类调用原有的方法，且对产生的结果进行控制

注：详见 http://www.cnblogs.com/malihe/p/6891920.html

### 7.(★★★)说明观察者模式的使用场景？为什么要使用观察者模式？

使用场景：

当一个抽象模型有两个方面，其中一个方面依赖于另一个方面。

当对一个对象的改变需要同时改变其它对象，而不知道具体有多少个对象待改变。

当一个对象必须通知其它对象，而它又不能假定其它对象是谁。换句话说，你不希望这些对象是紧密耦合的。

为什么要使用：

观察者模式符合接口隔离原则，实现了对象之间的松散耦合

一个对象的行为依赖于另一个对象的状态。或者换一种说法，当被观察对象（目标对象）的状态发生改变时 ，会直接影响到观察对象的行为。此时便需要用到观察者模式。被观察者和观察者一般是一对多的关系，一个被观察者 对应多个观察者，当一个被观察者的状态发生改变时，被观察者通知观察者，然后可以在观察者内部进行业务逻辑的处理。

### 8.(★★★)请写出您熟悉的几种设计模式，并做简单介绍

答：

**工厂方法模式**: 简单工厂模式有一个问题就是，类的创建依赖工厂类，也就是说，如果想要拓展程序，必须对工厂类进行修改，这违背了闭包原则，所以，从设计角度考虑，有一定的问题，如何解决？就用到工厂方法模式，创建一个工厂接口和创建多个工厂实现类，这样一旦需要增加新的功能，直接增加新的工厂类就可以了，不需要修改之前的代码

**单例模式**: 单例对象（Singleton）是一种常用的设计模式。在Java应用中，单例对象能保证在一个JVM中，该对象只有一个实例存在

**装饰器模式**: 装饰模式就是给一个对象增加一些新的功能，而且是动态的，要求装饰对象和被装饰对象实现同一个接口，装饰对象持有被装饰对象的实例

**代理模式**：其实每个模式名称就表明了该模式的作用，代理模式就是多一个代理类出来，替原对象进行一些操作，比如我们在租房子的时候回去找中介，为什么呢？因为你对该地区房屋的信息掌握的不够全面，希望找一个更熟悉的人去帮你做，此处的代理就是这个意思。再如我们有的时候打官司，我们需要请律师，因为律师在法律方面有专长，可以替我们进行操作，表达我们的想法。

https://blog.csdn.net/qq3965470/article/details/52167228

## 2. 框架

### 1.（★）EJB 与JAVA BEAN 的区别？

答：

Java Bean 是可复用的组件，对Java Bean 并没有严格的规范，理论上讲，任何一个Java 类都可以是一个Bean。但通常情况下，由于Java Bean 是被容器所创建（如Tomcat）的，所以Java Bean 应具有一个无参的构造器，另外，通常Java Bean 还要实现Serializable 接口用于实现Bean 的持久性。Java Bean实际上相当于微软COM 模型中的本地进程内COM 组件，它是不能被跨进程访问的。Enterprise Java Bean 相当于DCOM，即分布式组件。它是基于Java 的远程方法调用（RMI）技术的，所以EJB 可以被远程访问（跨进程、跨计算机）。但EJB必须被布署在诸如Webspere、WebLogic 这样的容器中，EJB 客户从不直接访问真正的EJB 组件，而是通过其容器访问。EJB 容器是EJB 组件的代理，EJB 组件由容器所创建和管理。客户通过容器来访问真正的EJB 组件。

### 2.（★）EJB 的几种类型？

答：

会话（Session）Bean、实体（Entity）Bean、消息驱动的（Message Driven）Bean；会话Bean 又可分为有状态（Stateful）和无状态（Stateless）两种；实体Bean可分为Bean 管理的持续性（BMP）和容器管理的持续性（CMP）两种。

### 3.（★）remote 接口和home 接口主要作用？

答：

remote 接口定义了业务方法，用于EJB 客户端调用业务方法；home 接口是EJB 工厂用于创建和移除查找EJB 实例。

### 4.（★）客服端口调用EJB 对象的几个基本步骤？

答：

设置JNDI 服务工厂以及JNDI 服务地址系统属性，查找Home 接口，从Home接口调用Create 方法创建Remote 接口，通过Remote 接口调用其业务方法。

### 5.（★★）EJB的角色和三个对象？

答：

一个完整的基于EJB 的分布式计算结构由六个角色组成，这六个角色可以由不同的开发商提供，每个角色所作的工作必须遵循Sun 公司提供的EJB 规范，以保证彼此之间的兼容性。这六个角色分别是EJB 组件开发者（Enterprise BeanProvider） 、应用组合者（Application Assembler）、部署者（Deployer） 、EJB 服务器提供者（EJB Server Provider）、EJB 容器提供者（EJB ContainerProvider）、系统管理员（System Administrator），这里面，EJB 容器是EJB之所以能够运行的核心。EJB 容器管理着EJB 的创建，撤消，激活，去活，与数据库的连接等等重要的核心工作；三个对象是Remote（Local）接口、Home（LocalHome）接口，Bean 类。

### 6.（★★）EJB 是基于哪些技术实现的？并说出SessionBean 和EntityBean 的区别，StatefulBean和StatelessBean的区别。

答：

EJB 包括Session Bean、Entity Bean、Message Driven Bean，基于JNDI、

RMI、JTA 等技术实现。

SessionBean 在J2EE 应用程序中被用来完成一些服务器端的业务操作，例如访问数据库、调用其他EJB 组件。EntityBean 被用来代表应用系统中用到的数据。

对于客户机，SessionBean 是一种非持久性对象，它实现某些在服务器上运行的业务逻辑。

对于客户机，EntityBean 是一种持久性对象，它代表一个存储在持久性存储器中的实体的对象视图，或是一个由现有企业应用程序实现的实体。

Session Bean 还可以再细分为Stateful Session Bean 与Stateless Session Bean ，这两种的Session Bean 都可以将系统逻辑放在method 之中执行，不同的是Stateful Session Bean 可以记录呼叫者的状态，因此通常来说，一个使用者会有一个相对应的Stateful Session Bean 的实体。Stateless Session Bean 虽然也是逻辑组件，但是他却不负责记录使用者状态，也就是说当使用者呼叫Stateless Session Bean 的时候，EJB Container 并不会找寻特定的Stateless Session Bean 的实体来执行这个method。换言之，很可能数个使用者在执行某个Stateless Session Bean 的methods 时，会是同一个Bean 的Instance 在执行。从内存方面来看， Stateful Session Bean 与Stateless Session Bean 比较， Stateful Session Bean 会消耗J2EE Server较多的内存，然而Stateful Session Bean 的优势却在于他可以维持使用者的状态。

### 7.（★★）bean实例的生命周期？

答：

对于Stateless Session Bean、Entity Bean、Message Driven Bean 一般存在缓冲池管理，而对于Entity Bean 和Statefull Session Bean 存在Cache管理，通常包含创建实例，设置上下文、创建EJB Object（create）、业务方法调用、remove 等过程，对于存在缓冲池管理的Bean，在create 之后实例并不从内存清除，而是采用缓冲池调度机制不断重用实例，而对于存在Cache 管理的Bean 则通过激活和去激活机制保持Bean 的状态并限制内存中实例数量。

### 8.（★★） EJB的激活机制？

答：

以Stateful Session Bean 为例：其Cache 大小决定了内存中可以同时存在的Bean 实例的数量，根据MRU 或NRU 算法，实例在激活和去激活状态之间迁移，激活机制是当客户端调用某个EJB 实例业务方法时，如果对应EJB Object发现自己没有绑定对应的Bean 实例则从其去激活Bean 存储中（通过序列化机制存储实例）回复（激活）此实例。状态变迁前会调用对应的ejbActive 和ejbPassivate 方法。

### 9.（★★） EJB包括（SessionBean,EntityBean）说出他们的生命周期，及如何管理事务的？

答：

SessionBean：Stateless Session Bean 的生命周期是由容器决定的，当客户机发出请求要建立一个Bean 的实例时，EJB 容器不一定要创建一个新的Bean的实例供客户机调用，而是随便找一个现有的实例提供给客户机。当客户机第一次调用一个Stateful Session Bean 时，容器必须立即在服务器中创建一个新的Bean 实例，并关联到客户机上，以后此客户机调用Stateful Session Bean 的方法时容器会把调用分派到与此客户机相关联的Bean 实例。EntityBean：EntityBeans 能存活相对较长的时间，并且状态是持续的。只要数据库中的数据存在，Entity beans 就一直存活。而不是按照应用程序或者服务进程来说的。即使EJB容器崩溃了，Entity beans 也是存活的。Entity Beans 生命周期能够被容器或者Beans 自己管理。EJB 通过以下技术管理事务：对象管理组织（OMG）的对象实务服务（OTS），Sun Microsystems 的Transaction Service（JTS）、JavaTransaction API（JTA），开发组（X/Open）的XA 接口。

### 10.（★★）EJB 的事务是如何实现的？何时进行回滚；

答：

是通过使用容器或Bean 自身管理事务的；当产生一个系统异常时容器就自动回滚事务。

### 11.（★★）EJB 容器提供的服务？

答:

主要提供生命周期管理、代码产生、持续性管理、安全、事务管理、锁和并发行管理等服务。

### 12.（★）EJB 需直接实现它的业务接口或Home 接口吗？请简述理由。

答：

远程接口和Home 接口不需要直接实现，他们的实现代码是由服务器产生的，程序运行中对应实现类会作为对应接口类型的实例被使用。

### 13.（★★）请对以下在J2EE 中常用的名词进行解释(或简单描述)

答：

web 容器：给处于其中的应用程序组件（JSP，SERVLET）提供一个环境，使JSP,SERVLET 直接跟容器中的环境变量接口交互，不必关注其它系统问题。主要由WEB 服务器来实现。例如：TOMCAT,WEBLOGIC,WEBSPHERE 等。该容器提供的接口严格遵守J2EE 规范中的WEB APPLICATION 标准。我们把遵守以上标准的WEB服务器就叫做J2EE 中的WEB 容器；EJB 容器：Enterprise java bean 容器。更具有行业领域特色。他提供给运行在其中的组件EJB 各种管理功能。只要满足J2EE 规范的EJB 放入该容器，马上就会被容器进行高效率的管理。并且可以通过现成的接口来获得系统级别的服务。例如邮件服务、事务管理；JNDI：（Java Naming & Directory Interface）JAVA 命名目录服务。主要提供的功能是：提供一个目录系统，让其它各地的应用程序在其上面留下自己的索引，从而满足快速查找和定位分布式应用程序的功能；JMS：（Java Message Service）JAVA 消息服务。主要实现各个应用程序之间的通讯。包括点对点和广播；

JTA：（ Java Transaction API）JAVA 事务服务。提供各种分布式事务服务。

应用程序只需调用其提供的接口即可；

JAF：（Java Action FrameWork）JAVA 安全认证框架。提供一些安全控制方面的框架。让开发者通过各种部署和自定义实现自己的个性安全控制策略；RMI/IIOP:（Remote Method Invocation /internet 对象请求中介协议）他们主要用于通过远程调用服务。例如，远程有一台计算机上运行一个程序，它提供股票分析服务，我们可以在本地计算机上实现对其直接调用。当然这是要通过一定的规范才能在异构的系统之间进行通信。RMI 是JAVA 特有的。

### 14.（★★）J2EE 是什么？

答：

J2EE 是Sun 公司提出的多层(multi-diered),分布式(distributed),基于组件(component-base)的企业级应用模型(enterpriese application model).在这样的一个应用系统中，可按照功能划分为不同的组件，这些组件又可在不同计算机上，并且处于相应的层次(tier)中。所属层次包括客户层(clietn tier)组件,web 层和组件,Business 层和组件,企业信息系统(EIS)层。

### 15.（★★）J2EE 是技术还是平台还是框架？

答：

J2EE 本身是一个标准，一个为企业分布式应用的开发提供的标准平台；J2EE 也是一个框架，包括JDBC、JNDI、RMI、JMS、EJB、JTA 等技术。

### 16.（★★）请写出spring 中I0C 的三种实现机制。

答：

三种机制为：通过setter 方法注入、通过构造方法注入和接口注入。

### 17.（★★）写出你熟悉的开源框架以及各自的作用。

答：

框架：hibernate、spring、struts；

Hibernate 主要用于数据持久化；

Spring 的控制反转能起到解耦合的作用；

Struts 主要用于流程控制。

### 18.（★★★）EJB 规范规定EJB 中禁止的操作有哪些？【较难】

答：

1)不能操作线程和线程API(线程API 指非线程对象的方法，如notify,wait等)；

2)不能操作awt；

3)不能实现服务器功能；

4)不能对静态属性存取；

5)不能使用IO 操作直接存取文件系统；

6)不能加载本地库；

7)不能将this 作为变量和返回；

8)不能循环调用。

### 19.（★★）什么是Spring框架？Spring框架有哪些主要模块？

答：

Spring框架是一个为Java应用程序的开发提供了综合、广泛的基础性支持的Java平台。Spring帮助开发者解决了开发中基础性的问题，使得开发人员可以专注于应用程序的开发。Spring框架本身亦是按照[设计模式](http://www.amazon.cn/gp/product/B001130JN8/ref=as_li_qf_sp_asin_il_tl?ie=UTF8&tag=importnew-23&linkCode=as2&camp=536&creative=3200&creativeASIN=B001130JN8)精心打造，这使得我们可以在开发环境中安心的集成Spring框架，不必担心Spring是如何在后台进行工作的。

Spring框架至今已集成了20多个模块。这些模块主要被分如下图所示的核心容器、数据访问/集成,、Web、AOP（面向切面编程）、工具、消息和测试模块。

**更多信息：**[Spring 框架教程](http://howtodoinjava.com/java-spring-framework-tutorials/)。

### 20.（★★）使用Spring框架能带来哪些好处？

答：

下面列举了一些使用Spring框架带来的主要好处：

* **Dependency Injection(DI)** 方法使得构造器和JavaBean properties文件中的依赖关系一目了然。
* 与EJB容器相比较，IoC容器更加趋向于轻量级。这样一来IoC容器在有限的内存和CPU资源的情况下进行应用程序的开发和发布就变得十分有利。
* Spring并没有闭门造车，Spring利用了已有的技术比如ORM框架、logging框架、J2EE、Quartz和JDK Timer，以及其他视图技术。
* Spring框架是按照模块的形式来组织的。由包和类的编号就可以看出其所属的模块，开发者仅仅需要选用他们需要的模块即可。
* 要测试一项用Spring开发的应用程序十分简单，因为测试相关的环境代码都已经囊括在框架中了。更加简单的是，利用JavaBean形式的POJO类，可以很方便的利用依赖注入来写入测试数据。
* Spring的Web框架亦是一个精心设计的Web [MVC框架](http://www.codeceo.com/article/mvc-framework-mvc-design.html)，为开发者们在web框架的选择上提供了一个除了主流框架比如Struts、过度设计的、不流行web框架的以外的有力选项。

Spring提供了一个便捷的事务管理接口，适用于小型的本地事物处理（比如在单DB的环境下）和复杂的共同事物处理（比如利用JTA的复杂DB环境）。

### 21.（★★★）什么是控制反转(IOC)？什么是依赖注入？

答：

控制反转是应用于软件工程领域中的，在运行时被装配器对象来绑定耦合对象的一种编程技巧，对象之间耦合关系在编译时通常是未知的。在传统的编程方式中，业务逻辑的流程是由应用程序中的早已被设定好关联关系的对象来决定的。在使用控制反转的情况下，业务逻辑的流程是由对象关系图来决定的，该对象关系图由装配器负责实例化，这种实现方式还可以将对象之间的关联关系的定义抽象化。而绑定的过程是通过“依赖注入”实现的。

控制反转是一种以给予应用程序中目标组件更多控制为目的设计范式，并在我们的实际工作中起到了有效的作用。

依赖注入是在编译阶段尚未知所需的功能是来自哪个的类的情况下，将其他对象所依赖的功能对象实例化的模式。这就需要一种机制用来激活相应的组件以提供特定的功能，所以依赖注入是控制反转的基础。否则如果在组件不受框架控制的情况下，框架又怎么知道要创建哪个组件？

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

### 22.（★★）请解释下Spring框架中的IoC？

答：

Spring中的 org.springframework.beans 包和 org.springframework.context包构成了Spring框架IoC容器的基础。

BeanFactory 接口提供了一个先进的配置机制，使得任何类型的对象的配置成为可能。ApplicationContex接口对BeanFactory（是一个子接口）进行了扩展，在BeanFactory的基础上添加了其他功能，比如与Spring的AOP更容易集成，也提供了处理message resource的机制（用于国际化）、事件传播以及应用层的特别配置，比如针对Web应用的WebApplicationContext。

**org.springframework.beans.factory.BeanFactory** 是Spring IoC容器的具体实现，用来包装和管理前面提到的各种bean。BeanFactory接口是Spring IoC 容器的核心接口。

### 23.（★★★）BeanFactory和ApplicationContext有什么区别？

答：

BeanFactory 可以理解为含有bean集合的工厂类。BeanFactory 包含了种bean的定义，以便在接收到客户端请求时将对应的bean实例化。

BeanFactory还能在实例化对象的时生成协作类之间的关系。此举将bean自身与bean客户端的配置中解放出来。BeanFactory还包含了bean生命周期的控制，调用客户端的初始化方法（initialization methods）和销毁方法（destruction methods）。

从表面上看，application context如同bean factory一样具有bean定义、bean关联关系的设置，根据请求分发bean的功能。但application context在此基础上还提供了其他的功能。

1. 提供了支持国际化的文本消息
2. 统一的资源文件读取方式
3. 已在监听器中注册的bean的事件

以下是三种较常见的 ApplicationContext 实现方式：

1、ClassPathXmlApplicationContext：从classpath的XML配置文件中读取上下文，并生成上下文定义。应用程序上下文从程序环境变量中取得。

ApplicationContext context = new ClassPathXmlApplicationContext(“bean.xml”);

2、FileSystemXmlApplicationContext ：由文件系统中的XML配置文件读取上下文。

ApplicationContext context = new FileSystemXmlApplicationContext(“bean.xml”);

3、XmlWebApplicationContext：由Web应用的XML文件读取上下文。

### 24.（★）Spring有几种配置方式？

答：

将Spring配置到应用开发中有以下三种方式：

1. 基于XML的配置
2. 基于注解的配置
3. 基于Java的配置

### 25.（★★）如何用基于XML配置的方式配置Spring？

答：

在Spring框架中，依赖和服务需要在专门的配置文件来实现，我常用的XML格式的配置文件。这些配置文件的格式通常用<beans>开头，然后一系列的bean定义和专门的应用配置选项组成。

SpringXML配置的主要目的时候是使所有的Spring组件都可以用xml文件的形式来进行配置。这意味着不会出现其他的Spring配置类型（比如声明的方式或基于[Java Class](http://www.codeceo.com/article/java-class.html)的配置方式）

Spring的XML配置方式是使用被Spring命名空间的所支持的一系列的XML标签来实现的。Spring有以下主要的命名空间：context、beans、jdbc、tx、aop、mvc和aso。

<beans>

<!-- JSON Support -->

<bean name="viewResolver" class="org.springframework.web.servlet.view.BeanNameViewResolver"/>

<bean name="jsonTemplate" class="org.springframework.web.servlet.view.json.MappingJackson2JsonView"/>

<bean id="restTemplate" class="org.springframework.web.client.RestTemplate"/>

</beans>

下面这个web.xml仅仅配置了**DispatcherServlet，**这件最简单的配置便能满足应用程序配置运行时组件的需求。

<web-app>

<display-name>Archetype Created Web Application</display-name>

<servlet>

<servlet-name>spring</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>spring</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

</web-app>

### 26.（★★）如何用基于Java配置的方式配置Spring？

答：

Spring对Java配置的支持是由@Configuration注解和@Bean注解来实现的。由@Bean注解的方法将会实例化、配置和初始化一个新对象，这个对象将由Spring的IoC容器来管理。@Bean声明所起到的作用与<bean/> 元素类似。被@Configuration所注解的类则表示这个类的主要目的是作为bean定义的资源。被@Configuration声明的类可以通过在同一个类的内部调用@bean方法来设置嵌入bean的依赖关系。

最简单的@Configuration 声明类请参考下面的代码：

@Configuration

public class AppConfig

{

@Bean

public MyService myService() {

return new MyServiceImpl();

}

}

对于上面的@Beans配置文件相同的XML配置文件如下：

<beans>

<bean id="myService" class="com.howtodoinjava.services.MyServiceImpl"/>

</beans>

上述配置方式的实例化方式如下：利用AnnotationConfigApplicationContext 类进行实例化

public static void main(String[] args) {

ApplicationContext ctx = new AnnotationConfigApplicationContext(AppConfig.class);

MyService myService = ctx.getBean(MyService.class);

myService.doStuff();

}

要使用组件组建扫描，仅需用@Configuration进行注解即可：

@Configuration

@ComponentScan(basePackages = "com.howtodoinjava")

public class AppConfig {

...

}

在上面的例子中，com.acme包首先会被扫到，然后再容器内查找被@Component 声明的类，找到后将这些类按照Sring bean定义进行注册。

如果你要在你的web应用开发中选用上述的配置的方式的话，需要用AnnotationConfigWebApplicationContext 类来读取配置文件，可以用来配置Spring的Servlet监听器ContrextLoaderListener或者Spring MVC的DispatcherServlet。

<web-app>

<!-- Configure ContextLoaderListener to use AnnotationConfigWebApplicationContext

instead of the default XmlWebApplicationContext -->

<context-param>

<param-name>contextClass</param-name>

<param-value>

org.springframework.web.context.support.AnnotationConfigWebApplicationContext

</param-value>

</context-param>

<!-- Configuration locations must consist of one or more comma- or space-delimited

fully-qualified @Configuration classes. Fully-qualified packages may also be

specified for component-scanning -->

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>com.howtodoinjava.AppConfig</param-value>

</context-param>

<!-- Bootstrap the root application context as usual using ContextLoaderListener -->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!-- Declare a Spring MVC DispatcherServlet as usual -->

<servlet>

<servlet-name>dispatcher</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- Configure DispatcherServlet to use AnnotationConfigWebApplicationContext

instead of the default XmlWebApplicationContext -->

<init-param>

<param-name>contextClass</param-name>

<param-value>

org.springframework.web.context.support.AnnotationConfigWebApplicationContext

</param-value>

</init-param>

<!-- Again, config locations must consist of one or more comma- or space-delimited

and fully-qualified @Configuration classes -->

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>com.howtodoinjava.web.MvcConfig</param-value>

</init-param>

</servlet>

<!-- map all requests for /app/\* to the dispatcher servlet -->

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

<url-pattern>/app/\*</url-pattern>

</servlet-mapping>

</web-app>

### 27.（★★）怎样用注解的方式配置Spring？

答：

Spring在2.5版本以后开始支持用注解的方式来配置依赖注入。可以用注解的方式来替代XML方式的bean描述，可以将bean描述转移到组件类的内部，只需要在相关类上、方法上或者字段声明上使用注解即可。注解注入将会被容器在XML注入之前被处理，所以后者会覆盖掉前者对于同一个属性的处理结果。

注解装配在Spring中是默认关闭的。所以需要在Spring文件中配置一下才能使用基于注解的装配模式。如果你想要在你的应用程序中使用关于注解的方法的话，请参考如下的配置。

<beans>

<context:annotation-config/>

<!-- bean definitions go here -->

</beans>

在 <context:annotation-config/>标签配置完成以后，就可以用注解的方式在Spring中向属性、方法和构造方法中自动装配变量。

下面是几种比较重要的注解类型：

1. **@Required：**该注解应用于设值方法。
2. **@Autowired：**该注解应用于有值设值方法、非设值方法、构造方法和变量。
3. **@Qualifier：**该注解和@Autowired注解搭配使用，用于消除特定bean自动装配的歧义。
4. **JSR-250 Annotations：**Spring支持基于JSR-250 注解的以下注解，@Resource、@PostConstruct 和 @PreDestroy。

### 28.（★★）请解释Spring Bean的生命周期？

答：

Spring Bean的生命周期简单易懂。在一个bean实例被初始化时，需要执行一系列的初始化操作以达到可用的状态。同样的，当一个bean不在被调用时需要进行相关的析构操作，并从bean容器中移除。

Spring bean factory 负责管理在spring容器中被创建的bean的生命周期。Bean的生命周期由两组回调（call back）方法组成。

1. 初始化之后调用的回调方法。
2. 销毁之前调用的回调方法。

Spring框架提供了以下四种方式来管理bean的生命周期事件：

* InitializingBean和DisposableBean回调接口
* 针对特殊行为的其他Aware接口
* Bean配置文件中的Custom init()方法和destroy()方法
* @PostConstruct和@PreDestroy注解方式

使用customInit()和 customDestroy()方法管理bean生命周期的代码样例如下：

<beans>

<bean id="demoBean" class="com.howtodoinjava.task.DemoBean"

init-method="customInit" destroy-method="customDestroy"></bean>

</beans>

更多内容请参考：Spring生命周期Spring Bean Life Cycle。

### 29.（★★）Spring Bean的作用域之间有什么区别？

答：

Spring容器中的bean可以分为5个范围。所有范围的名称都是自说明的，但是为了避免混淆，还是让我们来解释一下：

1. 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。
2. 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。
3. request：在请求bean范围内会每一个来自客户端的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：与请求范围类似，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：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。

**更多内容请参考 :** [Spring Bean Scopes](http://howtodoinjava.com/2013/05/07/spring-bean-scopes/)。

### 30.（★★）什么是Spring inner beans？

答：

在Spring框架中，无论何时bean被使用时，当仅被调用了一个属性。一个明智的做法是将这个bean声明为内部bean。内部bean可以用setter注入“属性”和构造方法注入“构造参数”的方式来实现。

比如，在我们的应用程序中，一个Customer类引用了一个Person类，我们的要做的是创建一个Person的实例，然后在Customer内部使用。

public class Customer

{

private Person person;

//Setters and Getters

}

public class Person

{

private String name;

private String address;

private int age;

//Setters and Getters

}

内部bean的声明方式如下：

<bean id="CustomerBean" class="com.howtodoinjava.common.Customer">

<property name="person">

<!-- This is inner bean -->

<bean class="com.howtodoinjava.common.Person">

<property name="name" value="lokesh" />

<property name="address" value="India" />

<property name="age" value="34" />

</bean>

</property>

</bean>

### 31.（★★）Spring框架中的单例Beans是线程安全的么？

答：

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**”。

### 32.（★★★）请举例说明如何在Spring中注入一个Java Collection？

答：

Spring提供了以下四种集合类的配置元素：

* **<list>** :   该标签用来装配可重复的list值。
* **<set>** :    该标签用来装配没有重复的set值。
* **<map>**:   该标签可用来注入键和值可以为任何类型的键值对。
* **<props>** : 该标签支持注入键和值都是字符串类型的键值对。

下面看一下具体的例子：

<beans>

<!-- Definition for javaCollection -->

<bean id="javaCollection" class="com.howtodoinjava.JavaCollection">

<!-- java.util.List -->

<property name="customList">

<list>

<value>INDIA</value>

<value>Pakistan</value>

<value>USA</value>

<value>UK</value>

</list>

</property>

<!-- java.util.Set -->

<property name="customSet">

<set>

<value>INDIA</value>

<value>Pakistan</value>

<value>USA</value>

<value>UK</value>

</set>

</property>

<!-- java.util.Map -->

<property name="customMap">

<map>

<entry key="1" value="INDIA"/>

<entry key="2" value="Pakistan"/>

<entry key="3" value="USA"/>

<entry key="4" value="UK"/>

</map>

</property>

<!-- java.util.Properties -->

<property name="customProperies">

<props>

<prop key="admin">admin@nospam.com</prop>

<prop key="support">support@nospam.com</prop>

</props>

</property>

</bean>

</beans>

### 33.（★★）如何向Spring Bean中注入一个Java.util.Properties？

答：

第一种方法是使用如下面代码所示的**<props>** 标签：

<bean id="adminUser" class="com.howtodoinjava.common.Customer">

<!-- java.util.Properties -->

<property name="emails">

<props>

<prop key="admin">admin@nospam.com</prop>

<prop key="support">support@nospam.com</prop>

</props>

</property>

</bean>

也可用”util:”命名空间来从properties文件中创建出一个propertiesbean，然后利用setter方法注入bean的引用。

### 34.（★★）请解释Spring Bean的自动装配？

答：

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。这意味着Spring可以通过向Bean Factory中注入的方式自动搞定bean之间的依赖关系。自动装配可以设置在每个bean上，也可以设定在特定的bean上。

下面的XML配置文件表明了如何根据名称将一个bean设置为自动装配：

<bean id="employeeDAO" class="com.howtodoinjava.EmployeeDAOImpl" autowire="byName" />

除了bean配置文件中提供的自动装配模式，还可以使用@Autowired注解来自动装配指定的bean。在使用@Autowired注解之前需要在按照如下的配置方式在Spring配置文件进行配置才可以使用。

<context:annotation-config />

也可以通过在配置文件中配置AutowiredAnnotationBeanPostProcessor 达到相同的效果。

<bean class ="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>

配置好以后就可以使用@Autowired来标注了。

@Autowired

public EmployeeDAOImpl ( EmployeeManager manager ) {

this.manager = manager;

}

### 35.（★★）请解释自动装配模式的区别？

答：

在Spring框架中共有5种自动装配，让我们逐一分析。

1. **no：**这是Spring框架的默认设置，在该设置下自动装配是关闭的，开发者需要自行在bean定义中用标签明确的设置依赖关系。
2. **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。
3. **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。
4. **constructor：**造器的自动装配和byType模式类似，但是仅仅适用于与有构造器相同参数的bean，如果在容器中没有找到与构造器参数类型一致的bean，那么将会抛出异常。

**autodetect：**该模式自动探测使用构造器自动装配或者byType自动装配。首先，首先会尝试找合适的带参数的构造器，如果找到的话就是用构造器自动装配，如果在bean内部没有找到相应的构造器或者是无参构造器，容器就会自动选择byTpe的自动装配方式。

### 36.（★★）如何开启基于注解的自动装配？

答：

要使用 @Autowired，需要注册 AutowiredAnnotationBeanPostProcessor，可以有以下两种方式来实现：

1、引入配置文件中的<bean>下引入 <context:annotation-config>

<beans>

<context:annotation-config />

</beans>

2、在bean配置文件中直接引入AutowiredAnnotationBeanPostProcessor

<beans>

<bean class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/>

</beans>

### 37.（★★）请举例解释@Required annotation？

答：

在产品级别的应用中，IoC容器可能声明了数十万了bean，bean与bean之间有着复杂的依赖关系。设值注解方法的短板之一就是验证所有的属性是否被注解是一项十分困难的操作。可以通过在<bean>中设置“**dependency-check**”来解决这个问题。

在应用程序的生命周期中，你可能不大愿意花时间在验证所有bean的属性是否按照上下文文件正确配置。或者你宁可验证某个bean的特定属性是否被正确的设置。即使是用“**dependency-check**”属性也不能很好的解决这个问题，在这种情况下，你需要使用@Required 注解。

需要用如下的方式使用来标明bean的设值方法。

public class EmployeeFactoryBean extends AbstractFactoryBean<Object>

{

private String designation;

public String getDesignation() {

return designation;

}

@Required

public void setDesignation(String designation) {

this.designation = designation;

}

//more code here

}

RequiredAnnotationBeanPostProcessor是Spring中的后置处理用来验证被@Required 注解的bean属性是否被正确的设置了。在使用RequiredAnnotationBeanPostProcesso来验证bean属性之前，首先要在IoC容器中对其进行注册：

<bean class="org.springframework.beans.factory.annotation.RequiredAnnotationBeanPostProcessor" />

但是如果没有属性被用 @Required 注解过的话，后置处理器会抛出一个BeanInitializationException 异常。

### 38.（★★）请举例解释@Autowired注解？

答：

@Autowired注解对自动装配何时何处被实现提供了更多细粒度的控制。@Autowired注解可以像@Required注解、构造器一样被用于在bean的设值方法上自动装配bean的属性，一个参数或者带有任意名称或带有多个参数的方法。

比如，可以在设值方法上使用@Autowired注解来替代配置文件中的 <property>元素。当Spring容器在setter方法上找到@Autowired注解时，会尝试用**byType** 自动装配。

当然我们也可以在构造方法上使用@Autowired 注解。带有@Autowired 注解的构造方法意味着在创建一个bean时将会被自动装配，即便在配置文件中使用<constructor-arg> 元素。

public class TextEditor {

private SpellChecker spellChecker;

@Autowired

public TextEditor(SpellChecker spellChecker){

System.out.println("Inside TextEditor constructor." );

this.spellChecker = spellChecker;

}

public void spellCheck(){

spellChecker.checkSpelling();

}

}

下面是没有构造参数的配置方式：

<beans>

<context:annotation-config/>

<!-- Definition for textEditor bean without constructor-arg -->

<bean id="textEditor" class="com.howtodoinjava.TextEditor">

</bean>

<!-- Definition for spellChecker bean -->

<bean id="spellChecker" class="com.howtodoinjava.SpellChecker">

</bean>

</beans>

### 39.（★★）请举例说明@Qualifier注解？

答：

@Qualifier注解意味着可以在被标注bean的字段上可以自动装配。Qualifier注解可以用来取消Spring不能取消的bean应用。

下面的示例将会在Customer的person属性中自动装配person的值。

public class Customer

{

@Autowired

private Person person;

}

下面我们要在配置文件中来配置Person类。

<bean id="customer" class="com.howtodoinjava.common.Customer" />

<bean id="personA" class="com.howtodoinjava.common.Person" >

<property name="name" value="lokesh" />

</bean>

<bean id="personB" class="com.howtodoinjava.common.Person" >

<property name="name" value="alex" />

</bean>

Spring会知道要自动装配哪个person bean么？不会的，但是运行上面的示例时，会抛出下面的异常：

Caused by: org.springframework.beans.factory.NoSuchBeanDefinitionException:

No unique bean of type [com.howtodoinjava.common.Person] is defined:

expected single matching bean but found 2: [personA, personB]

要解决上面的问题，需要使用 @Quanlifier注解来告诉Spring容器要装配哪个bean：

public class Customer

{

@Autowired

@Qualifier("personA")

private Person person;

}

### 40.（★★★）构造方法注入和设值注入有什么区别？

答：

请注意以下明显的区别：

1. 在设值注入方法支持大部分的依赖注入，如果我们仅需要注入int、string和long型的变量，我们不要用设值的方法注入。对于基本类型，如果我们没有注入的话，可以为基本类型设置默认值。在构造方法注入不支持大部分的依赖注入，因为在调用构造方法中必须传入正确的构造参数，否则的话为报错。
2. 设值注入不会重写构造方法的值。如果我们对同一个变量同时使用了构造方法注入又使用了设置方法注入的话，那么构造方法将不能覆盖由设值方法注入的值。很明显，因为构造方法尽在对象被创建时调用。
3. 在使用设值注入时有可能还不能保证某种依赖是否已经被注入，也就是说这时对象的依赖关系有可能是不完整的。而在另一种情况下，构造器注入则不允许生成依赖关系不完整的对象。
4. 在设值注入时如果对象A和对象B互相依赖，在创建对象A时Spring会抛出sObjectCurrentlyInCreationException异常，因为在B对象被创建之前A对象是不能被创建的，反之亦然。所以Spring用设值注入的方法解决了循环依赖的问题，因对象的设值方法是在对象被创建之前被调用的。

### 41.（★★★）Spring框架中有哪些不同类型的事件？

答：

Spring的ApplicationContext 提供了支持事件和代码中监听器的功能。

我们可以创建bean用来监听在ApplicationContext 中发布的事件。ApplicationEvent类和在ApplicationContext接口中处理的事件，如果一个bean实现了ApplicationListener接口，当一个ApplicationEvent 被发布以后，bean会自动被通知。

public class AllApplicationEventListener implements ApplicationListener < ApplicationEvent >

{

@Override

public void onApplicationEvent(ApplicationEvent applicationEvent)

{

//process event

}

}

Spring 提供了以下5中标准的事件：

1. **上下文更新事件（ContextRefreshedEvent）**：该事件会在ApplicationContext被初始化或者更新时发布。也可以在调用ConfigurableApplicationContext 接口中的refresh()方法时被触发。
2. **上下文开始事件（ContextStartedEvent）**：当容器调用ConfigurableApplicationContext的Start()方法开始/重新开始容器时触发该事件。
3. **上下文停止事件（ContextStoppedEvent）**：当容器调用ConfigurableApplicationContext的Stop()方法停止容器时触发该事件。
4. **上下文关闭事件（ContextClosedEvent）**：当ApplicationContext被关闭时触发该事件。容器被关闭时，其管理的所有单例Bean都被销毁。
5. **请求处理事件（RequestHandledEvent）**：在Web应用中，当一个http请求（request）结束触发该事件。

除了上面介绍的事件以外，还可以通过扩展ApplicationEvent 类来开发自定义的事件。

public class CustomApplicationEvent extends ApplicationEvent

{

public CustomApplicationEvent ( Object source, final String msg )

{

super(source);

System.out.println("Created a Custom event");

}

}

为了监听这个事件，还需要创建一个监听器：

public class CustomEventListener implements ApplicationListener < CustomApplicationEvent >

{

@Override

public void onApplicationEvent(CustomApplicationEvent applicationEvent) {

//handle event

}

}

之后通过applicationContext接口的publishEvent()方法来发布自定义事件。

CustomApplicationEvent customEvent = new CustomApplicationEvent(applicationContext, "Test message");

applicationContext.publishEvent(customEvent);

### 42.（★★）FileSystemResource和ClassPathResource有何区别？

答：

在FileSystemResource 中需要给出spring-config.xml文件在你项目中的相对路径或者绝对路径。在ClassPathResource中spring会在ClassPath中自动搜寻配置文件，所以要把ClassPathResource 文件放在ClassPath下。

如果将spring-config.xml保存在了src文件夹下的话，只需给出配置文件的名称即可，因为src文件夹是默认。

**简而言之，ClassPathResource在环境变量中读取配置文件，FileSystemResource在配置文件中读取配置文件。**

### 43.（★★★）Spring 框架中都用到了哪些设计模式？

答：

Spring框架中使用到了大量的设计模式，下面列举了比较有代表性的：

* 代理模式—在AOP和remoting中被用的比较多。
* 单例模式—在spring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. **RestTemplate**, JmsTemplate, JpaTemplate。
* 前端控制器—Srping提供了DispatcherServlet来对请求进行分发。
* 视图帮助(**View Helper** )—Spring提供了一系列的JSP标签，高效宏来辅助将分散的代码整合在视图里。
* 依赖注入—贯穿于BeanFactory / ApplicationContext接口的核心理念。
* 工厂模式—BeanFactory用来创建对象的实例。

### 44.（★★）IOC

答:

IoC就是Inversion of Control，控制反转。在Java开发中，IoC意味着将你设计好的类交给系统去控制，而不是在你的类内部控制。这称为控制反转。

下面我们以几个例子来说明什么是IoC。假设我们要设计一个Girl和一个Boy类，其中Girl有kiss方法，即Girl想要Kiss一个Boy。那么，我们的问题是，Girl如何能够认识这个Boy？

在我们中国，常见的ＭＭ与GG的认识方式有以下几种：1青梅竹马；2亲友介绍；3父母包办。 那么哪一种才是最好呢？青梅竹马：Girl从小就知道自己的Boy。

public class Girl ...{  InBlock  
ExpandedSubBlockStartContractedSubBlockvoid kiss()...{  
InBlockInBlockBoy boy = new Boy();  
InBlockExpandedSubBlockEnd}  
InBlockExpandedBlockEnd}

然而从开始就创建的Boy缺点就是无法在更换。并且要负责Boy的整个生命周期。如果我们的Girl想要换一个怎么办？（笔者严重不支持Girl经常更换Boy）亲友介绍：由中间人负责提供Boy来见面

public class Girl ...{  
InBlock  
ExpandedSubBlockStartContractedSubBlockvoid kiss()...{  
InBlock  
InBlockBoy boy = BoyFactory.createBoy();   
InBlockExpandedSubBlockEnd}InBlockExpandedBlockEnd}

亲友介绍，固然是好。如果不满意，尽管另外换一个好了。但是，亲友BoyFactory经常是以Singleton的形式出现，不然就是，存在于Globals，无处不在，无处不能。实在是太繁琐了一点，不够灵活。我为什么一定要这个亲友掺和进来呢？为什么一定要付给她介绍费呢？万一最好的朋友爱上了我的男朋友呢？   
  
父母包办：一切交给父母，自己不用费吹灰之力，只需要等着Kiss就好了。

ExpandedBlockStartContractedBlockpublic class Girl ...{  
InBlock  
ExpandedSubBlockStartContractedSubBlockvoid kiss(Boy boy)...{  
InBlock  
InBlock// kiss boy   
InBlock  
InBlockboy.kiss();  
InBlock  
ExpandedSubBlockEnd}  
InBlock  
ExpandedBlockEnd}

Well，这是对Girl最好的方法，只要想办法贿赂了Girl的父母，并把Boy交给他。那么我们就可以轻松的和Girl来Kiss了。看来几千年传统的父母之命还真是有用哦。至少Boy和Girl不用自己瞎忙乎了。  
  
这就是IOC，将对象的创建和获取提取到外部。由外部容器提供需要的组件。   
  
我们知道好莱坞原则：“Do not call us, we will call you.” 意思就是，You, girlie, do not call the boy. We will feed you a boy。   
  
我们还应该知道依赖倒转原则即 Dependence Inversion Princinple，DIP。Eric Gamma说，要面向抽象编程。面向接口编程是面向对象的核心。

组件应该分为两部分，即Service, 所提供功能的声明Implementation, Service的实现好处是：多实现可以任意切换，防止 “everything depends on everything” 问题．即具体依赖于具体。  
  
所以，我们的Boy应该是实现Kissable接口。这样一旦Girl不想kiss可恶的Boy的话，还可以kiss可爱的kitten和慈祥的grandmother。

### 45.（★★）IOC的type

答：

IoC的Type指的是Girl得到Boy的几种不同方式。我们逐一来说明。

IOC type 0：不用IOC

public class Girl implements Servicable ...{  
InBlockprivate Kissable kissable;  
public Girl() ...{  
InBlockkissable = new Boy();  
}  
public void kissYourKissable() ...{  
InBlockkissable.kiss();  
}  
}

Girl自己建立自己的Boy，很难更换，很难共享给别人，只能单独使用，并负责完全的生命周期。  
  
IOC type 1，先看代码：

public class Girl implements Servicable ...{  
Kissable kissable;  
public void service(ServiceManager mgr) ...{  
InBlockkissable = (Kissable) mgr.lookup(“kissable”);  
}  
InBlock  
public void kissYourKissable() ...{  
InBlockkissable.kiss();  
}  
}  
None

这种情况出现于Avalon Framework。一个组件实现了Servicable接口，就必须实现service方法，并传入一个ServiceManager。其中会含有需要的其它组件。只需要在service方法中初始化需要的Boy。另外，J2EE中从Context取得对象也属于type 1，它依赖于配置文件：   
  
                
       …   
  
  
IOC type 2：  
  
  
  
IOC type 3

public class Girl ...{  
InBlock  
InBlockprivate Kissable kissable;  
InBlock  
public void setKissable(Kissable kissable) ...{  
InBlock  
InBlockthis.kissable = kissable;  
InBlock  
ExpandedSubBlockEnd}  
InBlock  
public void kissYourKissable() ...{  
InBlock  
InBlockkissable.kiss();  
ExpandedSubBlockEnd}  
}

Type 2出现于Spring Framework，是通过JavaBean的set方法来将需要的Boy传递给Girl。它必须依赖于配置文件。

public class Girl ...{  
private Kissable kissable;  
public Girl(Kissable kissable) ...{  
this.kissable = kissable;  
}  
public void kissYourKissable() ...{  
InBlockkissable.kiss();  
}  
}

这就是PicoContainer的组件 。通过构造函数传递Boy给Girl。

NonePicoContainer container = new DefaultPicoContainer();  
None  
Nonecontainer.registerComponentImplementation(Boy.class);  
None  
Nonecontainer.registerComponentImplementation(Girl.class);  
None  
NoneGirl girl = (Girl) container.getComponentInstance(Girl.class);  
None  
Nonegirl.kissYourKissable();  
NoneNoneNone None

Well，以上的这些理论部分有些已经有了新的定义了。过些天我会再写一些文章具体说明。比如，原来的三种type结构现在已经重新定义为依赖注射的许多层次。  
  
IoC很年轻，还在发展。伴随着IOC的发展，AOP，COP，SOP等等都在不断的发展。作为程序员，随时关注着新的思想的发展是一件很轻松愉快的事情。有没有人愿意和我一起探讨学习共同进步呀！

### 46.（★★）Spring AOP面向方面编程原理：AOP概念(1)

答：

介绍完IoC之后，我们来介绍另外一个重要的概念：AOP(Aspect Oriented Programming)，也就是面向方面编程的技术。AOP基于IoC基础，是对OOP的有益补充。

AOP将应用系统分为两部分，核心业务逻辑（Core business concerns）及横向的通用逻辑，也就是所谓的方面Crosscutting enterprise concerns，例如，所有大中型应用都要涉及到的持久化管理（Persistent）、事务管理（Transaction Management）、安全管理（Security）、日志管理（Logging）和调试管理（Debugging）等。

AOP正在成为软件开发的下一个光环。使用AOP，你可以将处理aspect的代码注入主程序，通常主程序的主要目的并不在于处理这些aspect。AOP可以防止代码混乱。

Spring framework是很有前途的AOP技术。作为一种非侵略性的、轻型的AOP framework，你无需使用预编译器或其他的元标签，便可以在Java程序中使用它。这意味着开发团队里只需一人要对付AOP framework，其他人还是像往常一样编程。

### 47.（★★）AOP概念

答：

让我们从定义一些重要的AOP概念开始。

— 方面（Aspect）：一个关注点的模块化，这个关注点实现可能另外横切多个对象。事务管理是J2EE应用中一个很好的横切关注点例子。方面用Spring的Advisor或拦截器实现。

— 连接点（Joinpoint）：程序执行过程中明确的点，如方法的调用或特定的异常被抛出。

— 通知（Advice）：在特定的连接点，AOP框架执行的动作。各种类型的通知包括“around”、“before”和“throws”通知。通知类型将在下面讨论。许多AOP框架包括Spring都是以拦截器做通知模型，维护一个“围绕”连接点的拦截器链。

— 切入点（Pointcut）：指定一个通知将被引发的一系列连接点的集合。AOP框架必须允许开发者指定切入点，例如，使用正则表达式。

— 引入（Introduction）：添加方法或字段到被通知的类。Spring允许引入新的接口到任何被通知的对象。例如，你可以使用一个引入使任何对象实现IsModified接口，来简化缓存。— 目标对象（Target Object）：包含连接点的对象，也被称作被通知或被代理对象。

— AOP代理（AOP Proxy）：AOP框架创建的对象，包含通知。在Spring中，AOP代理可以是JDK动态代理或CGLIB代理。

— 编织（Weaving）：组装方面来创建一个被通知对象。这可以在编译时完成（例如使用AspectJ编译器），也可以在运行时完成。Spring和其他纯Java AOP框架一样，在运行时完成织入。

各种通知类型包括：

— Around通知：包围一个连接点的通知，如方法调用。这是最强大的通知。Aroud通知在方法调用前后完成自定义的行为，它们负责选择继续执行连接点或通过返回它们自己的返回值或抛出异常来短路执行。

— Before通知：在一个连接点之前执行的通知，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）。

— Throws通知：在方法抛出异常时执行的通知。Spring提供强制类型的Throws通知，因此你可以书写代码捕获感兴趣的异常（和它的子类），不需要从Throwable或Exception强制类型转换。

— After returning通知：在连接点正常完成后执行的通知，例如，一个方法正常返回，没有抛出异常。

Around通知是最通用的通知类型。大部分基于拦截的AOP框架（如Nanning和Jboss 4）只提供Around通知。

如同AspectJ，Spring提供所有类型的通知，我们推荐你使用最为合适的通知类型来实现需要的行为。例如，如果只是需要用一个方法的返回值来更新缓存，你最好实现一个after returning通知，而不是around通知，虽然around通知也能完成同样的事情。使用最合适的通知类型使编程模型变得简单，并能减少潜在错误。例如，你不需要调用在around通知中所需使用的MethodInvocation的proceed()方法，因此就调用失败。

切入点的概念是AOP的关键，它使AOP区别于其他使用拦截的技术。切入点使通知独立于OO的层次选定目标。例如，提供声明式事务管理的around通知可以被应用到跨越多个对象的一组方法上。 因此切入点构成了AOP的结构要素。

下面让我们实现一个Spring AOP的例子。在这个例子中，我们将实现一个before advice，这意味着advice的代码在被调用的public方法开始前被执行。以下是这个before advice的实现代码。

package com.ascenttech.springaop.test;

import java.lang.reflect.Method;

import org.springframework.aop.MethodBeforeAdvice;

public class TestBeforeAdvice implements MethodBeforeAdvice {

public void before(Method m, Object[] args, Object target)

throws Throwable {

System.out.println("Hello world! (by "

+ this.getClass().getName()

+ ")");

}

}

接口MethodBeforeAdvice只有一个方法before需要实现，它定义了advice的实现。before方法共用3个参数，它们提供了相当丰富的信息。参数Method m是advice开始后执行的方法，方法名称可以用作判断是否执行代码的条件。Object[] args是传给被调用的public方法的参数数组。当需要记日志时，参数args和被执行方法的名称都是非常有用的信息。你也可以改变传给m的参数，但要小心使用这个功能；编写最初主程序的程序员并不知道主程序可能会和传入参数的发生冲突。Object target是执行方法m对象的引用。

在下面的BeanImpl类中，每个public方法调用前，都会执行advice，代码如下。

package com.ascenttech.springaop.test;

public class BeanImpl implements Bean {

public void theMethod() {

System.out.println(this.getClass().getName()

+ "." + new Exception().getStackTrace()[0].getMethodName()

+ "()"

+ " says HELLO!");

}

}

类BeanImpl实现了下面的接口Bean，代码如下。

package com.ascenttech.springaop.test;

public interface Bean {

public void theMethod();

}

虽然不是必须使用接口，但面向接口而不是面向实现编程是良好的编程实践，Spring也鼓励这样做。

pointcut和advice通过配置文件来实现，因此，接下来你只需编写主方法的Java代码，代码如下。

package com.ascenttech.springaop.test;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.FileSystemXmlApplicationContext;

public class Main {

public static void main(String[] args) {

//Read the configuration file

ApplicationContext ctx

= new FileSystemXmlApplicationContext("springconfig.xml");

//Instantiate an object

Bean x = (Bean) ctx.getBean("bean");

//Execute the public method of the bean (the test)

x.theMethod();

}

}

我们从读入和处理配置文件开始，接下来马上要创建它。这个配置文件将作为粘合程序不同部分的“胶水”。读入和处理配置文件后，我们会得到一个创建工厂ctx，任何一个Spring管理的对象都必须通过这个工厂来创建。对象通过工厂创建后便可正常使用。

仅仅用配置文件便可把程序的每一部分组装起来，代码如下。

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN" "http://www.springframework. org/dtd/spring-beans.dtd">

<beans>

<!--CONFIG--

<bean id="bean" class="org.springframework.aop.framework.ProxyFactoryBean">

<property name="proxyInterfaces">

<value>com.ascenttech.springaop.test.Bean</value>

</property>

<property name="target">

<ref local="beanTarget"/>

</property>

<property name="interceptorNames">

<list>

<value>theAdvisor</value>

</list>

</property>

</bean>

<!--CLASS-->

<bean id="beanTarget" class="com.ascenttech.springaop.test.BeanImpl"/>

<!--ADVISOR--

<!--Note: An advisor assembles pointcut and advice-->

<bean id="theAdvisor" class="org.springframework.aop.support.RegexpMethod PointcutAdvisor">

<property name="advice">

<ref local="theBeforeAdvice"/>

</property>

<property name="pattern">

<value>com\.ascenttech\.springaop\.test\.Bean\.theMethod</value>

</property>

</bean>

<!--ADVICE-->

<bean id="theBeforeAdvice" class="com.ascenttech.springaop.test.TestBefore Advice"/>

</beans>

4个bean定义的次序并不重要。我们现在有了一个advice、一个包含了正则表达式pointcut的advisor、一个主程序类和一个配置好的接口，通过工厂ctx，这个接口返回自己本身实现的一个引用。

BeanImpl和TestBeforeAdvice都是直接配置。我们用一个惟一的ID创建一个bean元素，并指定了一个实现类，这就是全部的工作。

advisor通过Spring framework提供的一个RegexMethodPointcutAdvisor类来实现。我们用advisor的第一个属性来指定它所需的advice-bean，第二个属性则用正则表达式定义了pointcut，确保良好的性能和易读性。

最后配置的是bean，它可以通过一个工厂来创建。bean的定义看起来比实际上要复杂。bean是ProxyFactoryBean的一个实现，它是Spring framework的一部分。这个bean的行为通过以下的3个属性来定义。

— 属性proxyInterface定义了接口类。

— 属性target指向本地配置的一个bean，这个bean返回一个接口的实现。

— 属性interceptorNames是惟一允许定义一个值列表的属性，这个列表包含所有需要在beanTarget上执行的advisor。注意，advisor列表的次序是非常重要的。

### 48.（★）SpringMvc简单介绍下你对springMVC的理解?

答：

Spring MVC Framework有这样一些特点:

* 它是基于组件技术的.全部的应用对象,无论控制器和视图,还是业务对象之类的都是java组件.并且和Spring提供的其他基础结构紧密集成.
* 不依赖于Servlet API(目标虽是如此,但是在实现的时候确实是依赖于Servlet的)
* 可以任意使用各种视图技术,而不仅仅局限于JSP
* 支持各种请求资源的映射策略
* 它应是易于扩展的

### 49.（★）SpringMVC的工作流程?

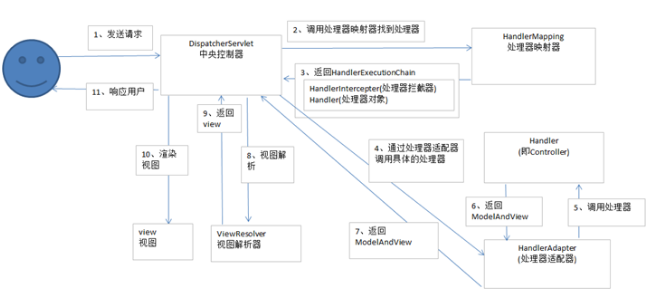
答：

1. 用户发送请求至前端控制器DispatcherServlet  
2. DispatcherServlet收到请求调用HandlerMapping处理器映射器。  
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通过HandlerAdapter处理器适配器调用处理器  
5. 执行处理器(Controller，也叫后端控制器)。  
6. Controller执行完成返回ModelAndView  
7. HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet  
8. DispatcherServlet将ModelAndView传给ViewReslover视图解析器  
9. ViewReslover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

### 50.（★）如果你也用过struts2.简单介绍下springMVC和struts2的区别有哪些?

1. springmvc的入口是一个servlet即前端控制器，而struts2入口是一个filter过虑器。  
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。

### 51.（★）SpringMVC执行流程



### 52.（★★）SSM优缺点、使用场景？

答：

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，不过mybatis可以通过XML或注解方式灵活配置要运行的sql语句，并将java对象和sql语句映射生成最终执行的sql，最后将sql执行的结果再映射生成java对象。

2. Mybatis学习门槛低，简单易学，程序员直接编写原生态sql，可严格控制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，例如互联网软件、企业运营类软件等，因为这类软件需求变化频繁，一但需求变化要求成果输出迅速。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（例如需求固定的定制化软件）如果用hibernate开发可以节省很多代码，提高效率。但是Hibernate的学习门槛高，要精通门槛更高，而且怎么设计O/R映射，在性能和对象模型之间如何权衡，以及怎样用好Hibernate需要具有很强的经验和能力才行。

4. 总之，按照用户的需求在有限的资源环境下只要能做出维护性、扩展性良好的软件架构都是好架构，所以框架只有适合才是最好。

### 53.（★）Spring有哪些优点？

轻量级：Spring在大小和透明性方面绝对属于轻量级的，基础版本的Spring框架大约只有2MB。

控制反转(IOC)：Spring使用控制反转技术实现了松耦合。依赖被注入到对象，而不是创建或寻找依赖对象。

面向切面编程(AOP)： Spring支持面向切面编程，同时把应用的业务逻辑与系统的服务分离开来。

容器：Spring包含并管理应用程序对象的配置及生命周期。

MVC框架：Spring的web框架是一个设计优良的web MVC框架，很好的取代了一些web框架。

事务管理：Spring对下至本地业务上至全局业务(JAT)提供了统一的事务管理接口。

异常处理：Spring提供一个方便的API将特定技术的异常(由JDBC, Hibernate, 或JDO抛出)转化为一致的、Unchecked异常。

### 54.（★）请你谈谈SSH整合：

答：

SSH： Struts（表示层）+Spring（业务层）+Hibernate（持久层）Struts：Struts是一个表示层框架，主要作用是界面展示，接收请求，分发请求。在MVC框架中，Struts属于VC层次，负责界面表现，负责MVC关系的分（View：沿 用JSP，HTTP，Form，Tag，Resourse ；Controller：ActionServlet，struts-config.xml，Action） Hibernate：

Hibernate是一个持久层框架，它只负责与关系数据库的操作。 Spring：

Spring是一个业务层框架，是一个整合的框架，能够很好地黏合表示层与持久层。

### 55.（★★）Spring里面applicationContext.xml文件能不能改成其他文件名？

答：

ContextLoaderListener是一个ServletContextListener, 它在你的web应用启动的时候初始化。缺省情况下，它会在WEB-INF/applicationContext.xml文件找Spring的配置。你可以通过定义一个<context-param>元素名字为”contextConfigLocation”来改变Spring配置文件的位置。示例如下：

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener <context-param>

<param-name>contextConfigLocation</param-name> <param-value>/WEB-INF/xyz.xml</param-value> </context-param></listener-class> </listener>

### 56.（★）如何在web应用里面配置spring?

答：

在web.xml中加入如下同容,

在启动web服务器时加载/WEB-INF/applicationContext.xml中的内容。

<servlet>

<servlet-name>context</servlet-name>

<servlet-class>

org.springframework.web.context.ContextLoaderServlet

</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

通过如下类得到ApplicationContext实例

WebApplicationContextUtils.getWebApplicationContext

### 57.（★）spring中的BeanFactory与ApplicationContext的作用有哪些？

答：

1. BeanFactory

负责读取bean配置文档，管理bean的加载，实例化，维护bean之间的依赖关系，负责bean的声明周期。

2. ApplicationContext除了提供上述BeanFactory所能提供的功能之外，还提供了更完整的框架功能：

a. 国际化支持

b. 资源访问：

Resource rs =ctx.getResource(”classpath:config.properties”),

“file:c:/config.properties”

c. 事件传递：通过实现ApplicationContextAware接口

### 58.（★）什么是aop，aop的作用是什么?

答：

面向切面编程（AOP）提供另外一种角度来思考程序结构，通过这种方式弥补了面向对象编程（OOP）的不足

除了类（classes）以外，AOP提供了切面。切面对关注点进行模块化，例如横切多个类型和对象的事务管理

Spring的一个关键的组件就是AOP框架，可以自由选择是否使用AOP

提供声明式企业服务，特别是为了替代EJB声明式服务。最重要的服务是声明性事务管理，这个服务建立在Spring的抽象事物管理之上

允许用户实现自定义切面，用AOP来完善OOP的使用 可以把Spring AOP看作是对Spring的一种增强

### 59.（★★）什么是Spring的依赖注入？有哪些不同类型的IOC（依赖注入）方式？

答：

依赖注入，是IOC的一个方面，是个通常的概念，它有多种解释。这概念是说你不用创建对象，而只需要描述它如何被创建。你不在代码里直接组装你的组件和服务，但是要在配置文件里描述哪些组件需要哪些服务，之后一个容器（IOC容器）负责把他们组装起来。

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。

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。

### 60.（★）在 Spring中如何注入一个java集合？

答：

Spring提供以下几种集合的配置元素：

<list>类型用于注入一列值，允许有相同的值。

<set> 类型用于注入一组值，不允许有相同的值。

<map> 类型用于注入一组键值对，键和值都可以为任意类型。

<props>类型用于注入一组键值对，键和值都只能为String类型。

### 61.（★）什么是Spring的MVC框架？

答：

Spring 配备构建Web 应用的全功能MVC框架。Spring可以很便捷地和其他MVC框架集成，如Struts，Spring 的MVC框架用控制反转把业务对象和控制逻辑清晰地隔离。它也允许以声明的方式把请求参数和业务对象绑定。

### 62.（★）DispatcherServlet

答：

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。

### 63.（★）关于spring说法错误的是（）

A.spring是一个轻量级JAVA EE的框架集合

B.spring是“依赖注入”模式的实现

C.使用spring可以实现声明事务

D.spring提供了AOP方式的日志系统

答: D

### 64.（★★）下面有关SPRING的事务传播特性，说法错误的是？

A.PROPAGATION\_SUPPORTS：支持当前事务，如果当前没有事务，就以非事务方式执行

B.PROPAGATION\_REQUIRED：支持当前事务，如果当前没有事务，就抛出异常

C.PROPAGATION\_REQUIRES\_NEW：新建事务，如果当前存在事务，把当前事务挂起

D.PROPAGATION\_NESTED：支持当前事务，新增Savepoint点，与当前事务同步提交或回滚

答: B

### 65.（★★）下列关于Spring特性中IoC描述错误的是：

A.IoC就是指程序之间的关系由程序代码直接操控

B.所谓“控制反转”是指控制权由应用代码转到外部容器，即控制权的转移

C.IoC将控制创建的职责搬进了框架中，从应用代码脱离开来

D.使用Spring的IoC容器时只需指出组件需要的对象，在运行时Spring的IoC容器会根据XML配置数据提供给它

答: A

### 66.（★★） 简述spring隔离级别与传播行为

答：

spring 的事务传播行为：

Spring在TransactionDefinition接口中规定了7种类型的事务传播行为，它们规定了事务方法和事务方法发生嵌套调用时事务如何进行传播：

PROPAGATION\_REQUIRED：如果当前没有事务，就新建一个事务，如果已经存在一个事务中，加入到这个事务中。这是最常见的选择。

PROPAGATION\_SUPPORTS：支持当前事务，如果当前没有事务，就以非事务方式执行。

PROPAGATION\_MANDATORY：使用当前的事务，如果当前没有事务，就抛出异常。

PROPAGATION\_REQUIRES\_NEW：新建事务，如果当前存在事务，把当前事务挂起。

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，则执行与PROPAGATION\_REQUIRED类似的操作。

Spring 的隔离级别

SERIALIZABLE：最严格的级别，事务串行执行，资源消耗最大；

REPEATABLE\_READ：保证了一个事务不会修改已经由另一个事务读取但未提交（回滚）的数据。避免了”脏读取”和”不可重复读取”的情况，但是带来了更多的性能损失。

READ\_COMMITTED:大多数主流数据库的默认事务等级，保证了一个事务不会读到另一个并行事务已修改但未提交的数据，避免了”脏读取”。该级别适用于大多数系统。

Read\_UNCOMMITTED：保证了读取过程中不会读取到非法数据

### 67.（★） 简述常见前后端服务框架

答：

SpringMVC,Spring,Mybatis,Dubbo,RabbitMQ,Redis

### 68.(★★★)框架书写

1. 请写出springMvc框架配置的主要部分

<!-- 前端控制器 -->

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- contextConfigLocation配置springmvc加载的配置文件(配置处理器映射器、适配器等等)

如果不配置contextConfigLocation，默认加载的是/WEB-INF/servlet名称-servlet.xml

-->

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:springmvc.xml</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

<!--

第一种：\*.action,访问以.action结尾由DispatcherServlet进行解析

第二种:/， 所有访问的地址都由DispatcherServlet进行解析，对于静态文件的解析需要配置不让DispatcherServlet进行解析

-->

<url-pattern>\*.action</url-pattern>

</servlet-mapping>

<!-- 处理器映射器 ，将bean的name作为URL进行查找,需要在配置Handler时指定beanName(就是URL)-->

<bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"></bean>

<!-- 处理器适配器，所有的处理器适配器都实现HandlerAdapter接口 -->

<bean class="org.springframework.web.servlet.mvc.SimpleControllerHandlerAdapter"></bean>

<!-- 配置视图解析器

解析jsp视图，默认使用jstl标签

CLASSPATH下面要有jstl jar包

-->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"></bean>

1. 请写出SSH或者SSM配置的主要部分

### 69.(★★★)ssh:

最主要的applicationContext.xml文件

[复制代码](javascript:void(0);)

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd

">

<!--1.数据源 c3p0-->

<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">

<property name="jdbcUrl" value="${jdbc.url}"></property>

<property name="driverClass" value="${jdbc.driverClassName}"></property>

<property name="user" value="${jdbc.username}"></property>

<property name="password" value="${jdbc.password}"></property>

</bean>

<!--2.识别到jdbc.properties-->

<context:property-placeholder location="classpath:jdbc.properties"></context:property-placeholder>

<!--3.形成SessionFactory-->

<bean id="sessionFactory" class="org.springframework.orm.hibernate5.LocalSessionFactoryBean">

<property name="dataSource" ref="dataSource"></property>

<property name="hibernateProperties">

<props>

<!--方言-->

<prop key="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</prop>

<!--是否打印sql-->

<prop key="hibernate.show\_sql">true</prop>

<prop key="hibernate.format\_sql">true</prop>

<prop key="hibernate.hbm2ddl.auto">update</prop>

<prop key="hibernate.current\_session\_context\_class">org.springframework.orm.hibernate5.SpringSessionContext</prop>

</props>

</property>

<!--关联小配置-->

<property name="mappingDirectoryLocations" value="classpath:cn/ssh/entity"></property>

</bean>

<!--4.dao-->

<bean id="deptDao" class="cn.ssh.dao.DeptDAOImpl">

<property name="sessionFactory" ref="sessionFactory"></property>

</bean>

<!--5.service-->

<bean id="deptService" class="cn.ssh.service.DeptServiceImpl">

<property name="dao" ref="deptDao"></property>

</bean>

<!--6.action-->

<bean id="deptAction" class="cn.ssh.action.DeptAction">

<property name="service" ref="deptService"></property>

</bean>

<!--7.事务管理器-->

<bean id="transactionManager" class="org.springframework.orm.hibernate5.HibernateTransactionManager">

<property name="sessionFactory" ref="sessionFactory"></property>

</bean>

<!--8.事务真实配置-->

<tx:annotation-driven transaction-manager="transactionManager"></tx:annotation-driven>

</beans>

#### Ssm

<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_4\_0.xsd"

version="4.0">

<!--全局范围内参数初始化-->

<context-param>

<param-name>contextConfigLocation</param-name> <!--参数名称-->

<param-value>classpath:applicationContext.xml</param-value> <!--参数取值-->

</context-param>

<context-param>

<param-name>AREACODE</param-name>

<param-value>410800000000</param-value>

</context-param>

<!--spring的监听器-->

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<!--springMVC的配置-->

<!--前端控制器-->

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<!-- 此处不配置 默认找 /WEB-INF/[servlet-name]-servlet.xml -->

<param-value>classpath:springmvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

<!--

1:\*.do \*.action 拦截以.do结尾的请求 (不拦截 jsp png jpg .js .css)

2:/ 拦截所有请求 (不拦截.jsp) 建议使用此种 方式 （拦截 .js.css .png) (放行静态资源)

3:/\* 拦截所有请求（包括.jsp) 此种方式 不建议使用

-->

<url-pattern>\*.do</url-pattern>

</servlet-mapping>

<!--<servlet>

<servlet-name>imageServlet</servlet-name>

<servlet-class>com.utils.PageAutocode</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>imageServlet</servlet-name>

<url-pattern>/servlet/imageServlet</url-pattern>

</servlet-mapping>-->

<!--如果某个会话在一定时间内未被访问，则服务器可以扔掉来节约内存 单位为分钟-->

<session-config>

<session-timeout>60</session-timeout>

</session-config>

<servlet>

<servlet-name>services</servlet-name>

<servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>services</servlet-name>

<url-pattern>/services/\*</url-pattern>

</servlet-mapping>

<!-- 静态资源文件的引入 -->

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.js</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.css</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>default</servlet-name>

<url-pattern>\*.png</url-pattern>

</servlet-mapping>

<!--过滤器-->

<filter>

<description>字符集过滤器</description>

<filter-name>encodingFilter</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<init-param>

<description>字符集编码</description>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>encodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<!--直接输入项目名进入时返回的页面-->

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

</web-app>

1. 请写出easyui或者bootstrap最常用控件的名字

#### Easyui

Pagination（分页）

Layout（布局）

Accordion（分类）

Tabs（选项卡）

LinkButton（按钮）

ComboBox（下拉列表框）

DateBox（日期输入框）

Slider（滑动条）

#### Bootstrap

表单 form-group

按钮 btn

按钮组 btn-toolbar

导航 nav

导航栏 nav-bar

面板 panel

表格 table

(4)请写出开发过程中最难的sql语句(mysql，Oracle均可）

## 3.杂项

### 3.1 XML

#### 1.（★）谈谈对XML的理解？说明Web应用中Web.xml文件的作用？

答：

可扩展标记语言，标准通用标记语言的子集，是一种用于标记电子文件使其具有结构性的标记语言。什么是可扩展标记语言？

可扩展标记语言是一种很像超文本标记语言的标记语言。

它的设计宗旨是传输数据，而不是显示数据。

它的标签没有被预定义。您需要自行定义标签。

它被设计为具有自我描述性。

它是W3C的推荐标准。

web.xml文件是用来配置：欢迎页、servlet、filter等的。当你的web工程没用到这些时，你可以不用web.xml文件来配置你的web工程。

#### 2.（★）写出你所知道的 XML 解析器

答：

DOM，SAX，SimpleXML，其中前两种是通用的解析器，和具体语言无关，而 SimpleXML 则是 PHP 提供的解析器。

1. **(★★)看下面的代码，说法正确的是(B)**

<bean id="userTable" class="com.xfaccp bean.UserTable">

<property name="userName">

<value>ACCP</value>

</property>

</bean>

A.其中<property name="userName">的userName是UserTable中的属性。可以不要get,set方法。

B.其中<property name="userName">的userName是UserTable中的属性，可以不要get方法，但是一定要有set方法

C.其中<property name="userName">的userName是UserTable中的属性，可以不要set方法，但是一定要有get方法

D.其中<property name="userName">的userName是UserTable 中的属性，定要有 get和set方法

答：B

### 3.2 编程题

#### 1.(★★)指出下面程序的运行结果

class A {

static {

System.out.print("1");

}

public A() {

System.out.print("2");

}

}

class B extends A{

static {

System.out.print("a");

}

public B() {

System.out.print("b");

}

}

public class Hello {

public static void main(String[] args) {

A ab = new B();

ab = new B();

}

}

结果：1a2b 2b

#### 2.(★★)JAVA实现调整数组顺序使奇数位于偶数前面问题

题目描述

输入一个整数数组，实现一个函数来调整该数组中数字的顺序 ,使得所有的奇数位于数组的前半部分,所有的偶数位于数组的后半部分，并保证奇数和奇数,偶数和偶数之间的相对位置不变。

public static void reOrderArray(int[] array) {

int len = array.length;

int[] arr = new int[len];

int j = 0;

for (int i = 0; i < len; i++) {

if (array[i] % 2 == 1) {

arr[j++] = array[i];

}

}

for (int i = 0; i < len; i++) {

if (array[i] % 2 == 0) {

arr[j++] = array[i];

}

}

for (int i = 0; i < len; i++) {

array[i]=arr[i];}

}

public static void reOrderArray2(int[] array) {

int len = array.length;

List ls=new ArrayList();

int j = 0;

for (int i = 0; i < len; i++) {

if (array[i] % 2 == 1) {

ls.add(array[i]);

}

}

for (int i = 0; i < len; i++) {

if (array[i] % 2 == 0) {

ls.add(array[i]);

}

}

for (int i = 0; i < len; i++) {

array[i]=(int) ls.get(i);}

}

#### 3.(★★)下面是People和Child类的定义和构造方法，每个构造方法都输出编号。在执行Child("mike")的时候输出结果是什么?

class People {

String nane;

public People(){

System.out.print(1);

}

public People(Sting name) {

System.out.print(2);

this.name = name;

}

}

class Child extends People {

People father:

public Child(String name) {

System.out.print(3);

this.name = name:

father = new People(name + ":F):

}

public Child() {

System.out.print(4);

}

}

结果：132

#### 4.(★★)指出下列程序的运行结果（）

Public class Example {

String str = new String (“good”);

char[] ch = (‘a’,’b’,’c’);

Public static void main (String args[]){

Example ex = new Example();

ex.change(ex.str,ex.ch);

System.out.print(ex.str+ “ and ”);

System.out.print(ex.ch);

}

Public void change(String str,char ch[]) {

Str = “test ok”;

Ch[0] = ‘g’;

}

}

结果：goodand gbc

#### 5.(★★)给定以下代码段

class A{

public A(){

System.out.println(“A”);

}

}

class B extends A{

public B(){

System.out.println(“B”)

}

public static void main(String[] args){

B b = new B();

}

}

上述代码将（B）

A、 不能通过编译

B、 通过编译，输出为B

C、 通过编译，输出为B

D、 通过编译，输出为A

#### 6.(★)下面程序的运行结果是（）为什么？

String str1 = “hello”;

String str2 = “he” + new String(“llo”);

System.err.println(str1 == str2);

答：运行结果：false；说明：String str1 = "hello";这里的str1指的是方法区的字符串常量池中的“hello”，编译时期就知道的； String str2 = "he" + new String("llo");这里的str2必须在运行时才知道str2是什么，所以它是指向的是堆里定义的字符串“hello”，所以这两个引用是不一样的。如果用str1.equal(str2)，那么返回的是True；因为两个字符串的内容一样。

#### 7.(★)如下代码输出结果是： A

int i = 1 , j = 10 ;

do{

if (i > --j){

continue;

}

} while(i++ < 5);

System.out.println(“ i =” + i + “and j = ” + j);

输出结果是什么

A. i = 6 and j = 5 B. i = 5 and j = 5 C. i = 6 and j = 6 D. i = 5 and j = 4

**8.(★)如下代码： B**

class Base{

Base(){ System.out.print(“Base”) ; }

}

public class Alpha extends Base{

public static void main(String[] args){

new Alpha();

new Base();

}

}

结果是什么：

A. Base B. BaseBase C.编译失败 D.代码运行但没有输出 E.运行时跑出异常

#### 9.(★)如下代码： A

public class Foo{

public static void main(String[] args){

try{

return;

} finally {

System.out.println(“Finally”) ;

}

}

}

输出结果是什么

A. Finally B.编译失败 C.代码运行但没有输出 D.运行时跑出异常

#### 10.(★)输出结果是什么A

int i = 0;

for(; i < 4 ; i += 2){

System.out.print( i + “”) ;

}

System.out.println( i ) ;

输出结果是什么

A. 0 2 4 B. 0 2 4 6 C. 0 1 2 3 4 D.编译失败 E.一个异常在运行时被抛出

#### 11.(★★)写出以下输出

class Parent {

private String p = "parent";

private static String sp ="s parent";

public void method(){

System.out.println(p);

}

public static void smethod(){

System.out.println(sp);

}

}

class Child extends Parent{

private String c = "child";

private static String sc = "s child";

public void method(){

System.out.println(c);

}

public static void smethod(){

System.out.println(sc);

}

}

public class T{

public static void main(String[] args) {

Child c = new Child();

Parent p = (Parent)c;

p.method();

p.smethod();

}

}

输出结果：

child

s parent

#### 12.(★)输出结果：

public class test{

public static void main(String[] args) {

StringBuffer a = new StringBuffer("A");

StringBuffer b = new StringBuffer("B");

operate(a ,b);

System.out.println(a +","+b);

}

public static void operate(StringBuffer x , StringBuffer y){

x.append(y);

y = x;

}

}

输出结果：

AB,B

#### 13.(★)下面程序运行的结果是（A）

public class Test{

public static void main(String[] args){

String str1 = “hello”;

String str2 = “he”+new String(“llo”);

System.err.println(str1 == str2);

}

}

A、false B、true C、Exception D、编译错误

#### 14.(★)有一个数组中有100个数字， 请输出:没有重复井按照升序排序的数字

public static void main(String[] args) {

Integer[] unsorteAndReapetArray = {2,1,58,4,2,36,6,5,4,85,6,2,5,4,6};

Set<Integer> treeSet = new TreeSet(Arrays.asList(unsorteAndReapetArray));

Integer[] sortedArray = treeSet.toArray(new Integer[]{});

System.out.println(Arrays.toString(sortedArray));

}

#### 15.(★)用最有效率的方法计算2乘以8？

2 << 3，因为将一个数左移n 位，就相当于乘以了2 的n 次方，那么，一个数乘以8 只要将其左移3 位即可，而位运算cpu 直接支持的，效率最高，所以，2 乘以8 等於几的最效率的方法是2 << 3。

#### 16.(★★)日期和时间

##### - 如何取得年月日、小时分钟秒？

//得到long类型当前时间

long l = System.currentTimeMillis();

//new日期对

Date date = new Date(l);

//转换提日期输出格式

SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-

dd HH:mm:ss");System.out.println(dateFormat.format(date));

##### - 如何取得从1970年1月1日0时0分0秒到现在的毫秒数？

//获取从1970 1 1到当前毫秒数

public class test1{

public static void main(String[] args) {

System.out.println(Calendar.getInstance().getTimeInMillis());

}

##### - 如何取得某月的最后一天？

Calendar calendar = Calendar.getInstance();

calendar.setTime(date);

calendar.set(Calendar.DAY\_OF\_MONTH, 1);

Date firstDayOfMonth = calendar.getTime();

calendar.add(Calendar.MONTH, 1);

calendar.add(Calendar.DAY\_OF\_MONTH, -1);

Date lastDayOfMonth = calendar.getTime();

##### - 如何格式化日期？

Date date=new Date();

SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

formatter.format(date);

#### 17.(★★)给定整数序列A1，A2，... An(可能有负数),求A1-An的一个子序列Ai-Aj，使得Ai到Aj的和最大，并输出子序列内容

import java.util.Scanner;

public class pptv\_bishi {

//双重循环法求解，时间复杂度O(n\*3)。

public static void main(String[] args) {

int n;

System.out.println("请输入序列长度n:");

Scanner in=new Scanner(System.in);

n=in.nextInt();

System.out.println("请输入n个序列值:");

int [] arr=new int[n+1];

for(int i=1;i<=n;i++){//输入序列An。

arr[i]=in.nextInt();

}

in.close();

double sum=0;//记录最大子序列和。

double temp=0;

int low=0,high=0;

for(int i=1;i<=n;i++){

for(int j=i;j<=n;j++){

temp=doSearch(i,j,arr);

if(temp>sum){

sum=temp;

low=i;

high=j;

}

}

}

System.out.println("the substring is: \nSTART..."+low+"\nEND....."+high+"\nMaxSum is "+sum);

}

private static double doSearch(int low,int high,int[] arr) {

double value=0;

for(int i=low;i<=high;i++){

value+=arr[i];

}

return value;

}

}

#### 18.(★★★)类ExampleA继承Exception，类ExampleB继承ExampleA

有如下代码片断：

|  |
| --- |
| try {      throw new ExampleB("b")  } catch（ExampleA e）{      System.out.println("ExampleA");  } catch（Exception e）{      System.out.println("Exception");  } |

请问执行此段代码的输出是什么？

答：输出：ExampleA。（根据里氏代换原则[能使用父类型的地方一定能使用子类型]，抓取ExampleA类型异常的catch块能够抓住try块中抛出的ExampleB类型的异常）

#### 19.(★★)写一个方法，输入一个文件名和一个字符串，统计这个字符串在这个文件中出现的次数。

答：

/\*\*

\* 写入一个方法，输入一个文件名和一个字符串，统计这个字符串在这个文件中出现的次数。

\* @param fileName 文件名

\* @param str 查找的字符串

\* @return

\* @throws Exception

\*/

//方法一

public static int funCount1(String fileName,String str) throws Exception {

int count = 0;

BufferedReader bf = new BufferedReader(new InputStreamReader(new FileInputStream(fileName)));

String line ;

StringBuilder sb = new StringBuilder();

while((line = bf.readLine() )!= null) {

sb = sb.append(line);

}

int a = 0;

while((a = sb.indexOf(str)) != -1) {

sb = sb.delete(a, a + str.length());

count++;

}

return count;

}

//方法二：正则表达式

public static int funCount2(String fileName,String str) throws Exception {

int count =0 ;

BufferedReader bf = new BufferedReader(new InputStreamReader(new FileInputStream(fileName)));

String line ;

StringBuilder sb = new StringBuilder();

while((line = bf.readLine() )!= null) {

sb = sb.append(line);

}

String pattern = ".\*" + str + ".\*";

while(Pattern.matches(pattern, sb.toString())) {

count ++;

int a = sb.indexOf(str);

sb.delete(a, a + str.length());

}

return count;

}

#### 20.(★★)如何用Java代码列出一个目录下所有的文件？

答：

File file=new File("H:\\");

for(File temp:file.listFiles()){//Java5的新特性之一就是增强的for循环。

上面的for循环的意思是：定义一个File的变量temp，变量child会自动递增遍历File类型的数组listFiles

我们不再需要写得像原来那么复杂了，数组、迭代器都可以这样使用，

if(temp.isDirectory()){

System.out.println(temp.toString());

}

}

结果：

H:\$RECYCLE.BIN

H:\360CloudUI

H:\BaiduYunDownload

H:\Hlddz

H:\QQMusicCache

H:\System Volume Information

#### 21.(★★★)用Java的套接字编程实现一个多线程的回显（echo）服务器。

答：

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

import java.net.ServerSocket;

import java.net.Socket;

public class EchoServer {

private static final int ECHO\_SERVER\_PORT = 6789;

public static void main(String[] args) {

try(ServerSocket server = new ServerSocket(ECHO\_SERVER\_PORT)) {

System.out.println("服务器已经启动...");

while(true) {

Socket client = server.accept();

new Thread(new ClientHandler(client)).start();

}

} catch (IOException e) {

e.printStackTrace();

}

}

private static class ClientHandler implements Runnable {

private Socket client;

public ClientHandler(Socket client) {

this.client = client;

}

@Override

public void run() {

try(BufferedReader br = new BufferedReader(new InputStreamReader(client.getInputStream()));

PrintWriter pw = new PrintWriter(client.getOutputStream())) {

String msg = br.readLine();

System.out.println("收到" + client.getInetAddress() + "发送的: " + msg);

pw.println(msg);

pw.flush();

} catch(Exception ex) {

ex.printStackTrace();

} finally {

try {

client.close();

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

}

#### 22.(★)用java编写一个桌子类，类中有三个属性:长，宽，高，一个方法，方法的功能为获得桌子的体积。

答：

public class table {

double length;

double width;

double height;

public double getVolume(int length,int width,int height){

double volume=length\*width\*height;

return volume;

}

}

#### 23.(★)写一段简单的代码。遍历一个HashMap

答：

Map map=**new** HashMap();

Set set = map.keySet();  
**for** (Object a: set) {  
 System.***out***.println(map.get(a));  
}

#### 24.(★★)实现如下函数

函数功能：判断一个字符串是否为一个合法的ipv4地址，如：10.34.100.9 一个合法的ipv4地址分为4段，用逗号分开，每段的值在0到255之间

函数输入：任意字符串

函数输出：true代表是一个合法的IPV4地址，false非法

要求：1必须用java代码完整实现2不能使用正则表达式

答：

**public boolean** isipv4(String str){  
 String[] split = str.split(**"\\."**);  
 **if** (split.**length**==4){  
 **int** a=Integer.*parseInt*(split[0]);  
 **int** b=Integer.*parseInt*(split[1]);  
 **int** c=Integer.*parseInt*(split[2]);  
 **int** d=Integer.*parseInt*(split[3]);  
 **if** (a>=0 && a<=255 && b>=0 && b<=255 && c>=0 && c<=255 && d>=0 && d<=255) {  
 **return true**;  
 }**else return false**;  
 }**else return false**;  
}

#### 25.(★)下面程序，加入哪段代码可以正常运行

Public class ExceptionTest {

Class TestExceptionTest extends Exception{}

public void runTest()/\*加入代码区域\*/{

runTest();

}

}

1. throws Exception B.catch (Exception e)

答：

A

#### 26.(★)下面程序输出什么?

public class X{

public static void main(String args[]){

Object o1=new Object();

Object o2=o1;

if(o1.equals(o2)){

System.out println(“Equal”);

}

}

}

答：

Equal

#### 27.(★★)编程实现大整数加法，例如c=a+b，String a.size 大于1000位

public static String bigNumberSumBetter(String num1, String num2) {

char sign = '+';

char sign1 = num1.charAt(0);

char sign2 = num2.charAt(0);

String number1 = "";

String number2 = "";

// 去符号位操作

if (sign1 == '-' || sign1 == '+') {

number1 = num1.substring(1);

} else {

sign1 = '+';

number1 = num1;

}

// 去符号位操作

if (sign2 == '-' || sign2 == '+') {

number2 = num2.substring(1);

} else {

sign2 = '+';

number2 = num2;

}

boolean isDig1 = number1.matches("[1-9][0-9]\*");

boolean isDig2 = number2.matches("[1-9][0-9]\*");

if (!isDig1 || !isDig2) {

throw new NumberFormatException("输入的数据不是正确的格式的整数");

}

//两个数的长度

int length1 = number1.length();

int length2 = number2.length();

int len = length1>=length2? length1+1:length2+1;

StringBuffer number1Buffer = new StringBuffer();

StringBuffer number2Buffer = new StringBuffer();

//扩展数据的长度,使它们的长度一样

if(length1>length2){

for(int i=0; i<length1-length2; i++){

number2Buffer.append("0");

}

}else if(length1<length2){

for(int i=0; i<length2-length1; i++){

number1Buffer.append("0");

}

}

number1Buffer.append(number1);

number2Buffer.append(number2);

char[] chars1 = number1Buffer.reverse().toString().toCharArray();

char[] chars2 = number2Buffer.reverse().toString().toCharArray();

//存储每位相加的结果

int[] result = new int[len];

//同号相加

if(sign1==sign2){

sign = sign1;

for(int i=0; i<len-1; i++){

result[i] = (chars1[i]-'0')+(chars2[i]-'0');

}

// 处理进位

for (int i = 0; i < len; i++) {

if (result[i] >= 10) {

result[i + 1] += result[i] / 10;

result[i] = result[i] % 10;

}

}

}else {

// 拿大的数减去小的数

boolean lager = number1.compareTo(number2)>0 ? true : false;

if (lager) {

sign = sign1;

for (int i = 0; i < len-1; i++) {

result[i] = (chars1[i] - '0') - (chars2[i] - '0');

}

} else {

sign = sign2;

for (int i = 0; i < len-1; i++) {

result[i] = (chars2[i] - '0') - (chars1[i] - '0');

}

}

// 处理借位

for (int i = 0; i < len; i++) {

if (result[i] < 0) {

result[i] += 10;

result[i + 1]--;

}

}

}

// 结果没有进位时的0处理

boolean flag = true;

StringBuffer resultStr = new StringBuffer();

for (int i = result.length - 1; i >= 0; i--) {

if (result[i] == 0 && flag) {

continue;

}

flag = false;

resultStr.append(result[i]);

}

// 符号处理

if (sign == '-') {

return "-" + resultStr.toString();

} else {

return resultStr.toString();

}

}

#### 28. (★★)请写一个递归调用的java函数，输入一个字符串，输出的是这个字符串里包含数字的个数，如“487449023”里包含3个4

import java.util.Collection;

import java.util.HashMap;

import java.util.Scanner;

import java.util.Set;

public class Text {

//盛放数据的map

static HashMap<Character,Integer> hashMap = new HashMap<>();

//程序核心区

public static void getWordCount(String str){

if(str.length()>=1){

//每次取出最后一个字符

char singleChar = str.charAt(str.length() - 1);

//留下剩下的字符串继续递归调用

String newString = str.substring(0, str.length() - 1);

if(singleChar>=48 && singleChar<=57) {

//第一次符合数字，添加到map

if(hashMap.get(singleChar)==null) {

hashMap.put(singleChar,1);

} else {

//每次出现增加出现次数

hashMap.put(singleChar,hashMap.get(singleChar)+1);

}

}

getWordCount(newString);

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String input = scanner.next();

//递归调用方法

getWordCount(input);

//遍历输出结果

Set<Character> characters = hashMap.keySet();

for (Character character : characters) {

System.out.println(character+","+hashMap.get(character));

}

}

}

#### 29.(★★)编程实现：输入n，依次输入1到最大的n位数，例如：输入3，打印1,2,3...,999.

System.out.print("请输入n");

Scanner scan = new Scanner(System.in);

int read = scan.nextInt();

String digits="";

for(int i=1;i<=read;i++){

digits=digits+"9";

}

int maxNum = Integer.parseInt(digits);

for (int i=1;i<=maxNum;i++){

System.out.print(i+",");

}

#### 30.(★)在字符串中找出第一个只出现一次的字符。如输入“abaccdeff”，则输出’b’.

System.out.print("请输入字符串");

Scanner scan = new Scanner(System.in);

String str = scan.nextLine();

for (int i = 0; i < str.length(); i++) {

//截去第i个字符

String temp = str.substring(0, i) + str.substring(i + 1);

//在剩余的字符串中搜索有没有与第i个字符相同的，没有的时候会返回-1

int idx = temp.indexOf(str.charAt(i));

//剩下的字符串中没有的，就是唯一的

if (idx == -1) {

String c = String.valueOf(str.charAt(i));

System.out.println("第一个不重复字符是"+c);

break;

}

}

System.out.print("请输入字符串");

Scanner scan = new Scanner(System.in);

String str = scan.nextLine();

for (int i = 0; i < str.length(); i++) {

//截去第i个字符

String temp = str.substring(0, i) + str.substring(i + 1);

//在剩余的字符串中搜索有没有与第i个字符相同的，没有的时候会返回-1

int idx = temp.indexOf(str.charAt(i));

//剩下的字符串中没有的，就是唯一的

if (idx == -1) {

String c = String.valueOf(str.charAt(i));

System.out.println("第一个不重复字符是"+c);

break;

}

}

#### 31.(★)写一个熟悉的排序，如6,4,3,5，1,8,12,34,9，请按照从小到大输出

int str[]={6,4,3,5,1,8,12,34,9};

for (int i=1;i<str.length;i++){

for(int j=0;j<=str.length-i-1;j++){

if(str[j]>str[j+1]){

int t=str[j];

str[j]=str[j+1];

str[j+1]=t;

}

}

}

for (int i:str) {

System.out.print(i+",");

}

#### 32.(★)写出一个singleton（可以写伪码）

饿汉式（线程安全）：

public class Singleton{

private static Singleton instance = new Singleton();

//私有构造方法

private Singleton(){}

//静态方法为调用者提供单利对象

public static Singleton getInstance(){

return instance;

}

}

懒汉式（不安全）：

public class Singleton

{

private static Singleton instance;

private Singleton(){}

private static Singleton getInstnce()

{

if(instance == null)

{

instance = new Singleton();

}

return instance;

}

}

懒汉式（安全）：

public class Singleton

{

private static Singleton instance;

private Singleton(){}

private static synchronized Singleton getInstnce()

{

if(instance == null)

{

instance = new Singleton();

}

return instance;

}

}

#### 33.(★★)Java去除文件中的重复行

几天在做数据校正的时候，其中一个流程需要在几千行数据中剔除其中重复的数据，这个工作量就比较大了，虽然excel也有去除重复行的功能，但想到IO流中有读取一行数据的方法，如果用程序实现，应该要比用excel处理快得多，毕竟excel还要来回复制粘贴，而程序只要run一下就好了，下面就是具体实现代码

public static void main(String[] args) throws Exception {

//需要处理数据的文件位置

FileReader fileReader = new FileReader(new File(“E:\\tt.txt”));

BufferReadre bufferReader = new BufferReader(fileReader);

Map<String,String> map = new HashMap<String,String>();

String readLine = null;

int i = 0;

while((readLine = bufferReader.readLine()) != null) {

//每次读取一行数据，与map进行比较，如果该行数据map中没有，就保存到map集合中

if(!map.containsValue(readLine)) {

map.put(“key” + i,readLine);

i++;

}

}

for(int j = 0;j < map.size();j++) {

System.out.println(map.get(“key” + j)):

}

}

对于要给有序数组。我我们通常采用二分查找的方式来定位某一元素，请编写二分查找的算法，在数组查找指定的元素。

答：代码如下：

/\*\*

\* @description 二分法查找指定元素位置

\*\*/

public class Solution {

public static int MyBinarySearch(int[] arr , int key){

//先创建两个索引值,前边的索引和后边的索引,来规定查找的界限

int before = 0;

int back = arr.length-1;

while(back >before){

//创建一个中间值索引

int mid = (back+before)/2;

if(arr[mid] > key){

back = mid-1;

}else if(arr[mid] < key){

before = mid+1;

}else{

return mid;

}

}

return back;

}

public static void main(String[] args) {

int[] arr = {3,2,5,9,7};

//返回指定数组元素的下标

System.out.println(MyBinarySearch(arr,5));

}

}

#### 34.(★★)对于一个字符串。请设计一个高效算法。找到第一次重复出现的字符。

答：代码如下：

/\*\*

\* @description 查找指定字符串第一次重复出现的字符

\*\*/

public class FirstRepeatDemo {

public static String findFirstRepeatChar(String string) {

int[] charset = new int[8];

char[] chars = string.toCharArray();

for (char c : chars) {

int row = c / 32;

int col = c % 32;

if ((charset[row] & 1 << col) != 0) {

return String.valueOf(c);

} else {

charset[row] |= (1 << col);

}

}

return "-1";

}

public static void main(String[] args) {

//结果：y

String result = findFirstRepeatChar("qywyer23tdd");

System.out.println(result);

}

}

#### 35.(★★)JAVA实现一种排序

答：https://www.cnblogs.com/0201zcr/p/4763806.html

#### 36.(★★)java实现冒泡排序

答：代码如下：

/\*

\* 冒泡排序

\*/

public class BubbleSort {

　　public static void main(String[] args) {

　　　　int[] arr={6,3,8,2,9,1};

　　　　System.out.println("排序前数组为：");

　　　　for(int num:arr){

　　　　　　System.out.print(num+" ");

　　　　}

　　　　for(int i=0;i<arr.length-1;i++){//外层循环控制排序趟数

　　　　　　for(int j=0;j<arr.length-1-i;j++){//内层循环控制每一趟排序多少次

　　　　　　　　if(arr[j]>arr[j+1]){

　　　　　　　　　　int temp=arr[j];

　　　　　　　　　　arr[j]=arr[j+1];

　　　　　　　　　　arr[j+1]=temp;

　　　　　　　　}

　　　　　　}

　　　　}

　　　　System.out.println();

　　　　System.out.println("排序后的数组为：");

　　　　for(int num:arr){

　　　　　　System.out.print(num+" ");

　　　　}

　　}

}

#### 37.(★★)打印出1000以内的“水仙花数“，所谓”水仙花数“是指一个三位数字立方和等于该数本身。例如：153是一个”水仙花数“，因为153=1的三次方+5的三次方+3的三次方

答：代码如下：

方法1：

public class Test {

public static void main(String[] args) {

int i,num,j,k,sum;

int[] a=new int[5];

System.out.println("1000以内的水仙花数为：");

for(i=0;i<=1000;i++){

num=i;

j=0;

sum=0;

while(num!=0){

a[j]=num%10;

num=num/10;

j++;

}

for(k=0;k<j;k++) {

sum=(int) (sum+Math.pow(a[k], j));

}

if(sum==i)

System.out.print(i+" ");

}

}

}

#### 38.(★★★)将‘桌面/面试/src‘文件夹下文件数据格式转换为’桌面/面试/result‘文件夹下的数据

源文件数据说明：

1. 首行为标题行

time = 时间

word = 词

appId = 应用id

pos = 应用对应词位置

其他参数无用

结果文件数据说明：

a． 文件名为appId

b． 首行为标题行

例如：

time word1 word2 word3

2017-09-07 22 pos1 pos2

time：时间

word1、word2、word3：对应源文件的word

pos1、pos2、：对应源文件pos，词下没有对应位置则为空

例子

源格式

time word appId pos count hot

2017-08-12 21 1点新闻 01183239906 659 2160 4615

结果格式

time 1点新闻

2017-08-12 659

目标：实现QpsController.java中的control方法，使得Main.java中main方法可以正常运行。要求：不能改写Main.java(threadSize和qpsLimit参数可以任意调整)和Worker.java.不得删除任何已有代码，可以增加个人认为必要的其他java类，QpsController.java可以随意增加代码

package com.aliyun.ezwork.test;

/\*\*

\* Main

\*

\*/

public class Main {

public static void main(String[] args) throws InterruptedException {

//线程数和qps值都不要设置过大，避免系统性能差别导致测试断言不准建议线程数控制在2位数 qps控制在3位数

int threadSize = 5;

int qpsLimit = 8;

QpsController controller = new QpsController(qpsLimit);

for(int i = 0 ; i < threadSize ; i++){

Thread t = new Thread(){

Worker worker = new Worker();

@Override

public void run() {

while(true){

controller.control();

worker.work();

}

}

};

t.setName("worker-threads-" + i);

t.start();

}

int loopCount = 1;

while(true){

//在循环稳定运行几轮之后 观察worker.getcount和except的值是否相等 相等表示限制已经开始生效

//第一轮循环中 work线程和主线程之间由于竞争关系 可能会出现count值和except值不相等的情况

System.out.println(Worker.getCount() + " -- except " + (qpsLimit \* loopCount));

Thread.sleep(1000);

loopCount++;

}

}

}

package com.aliyun.ezwork.test;

/\*\*

\* QpsController

\*

\*/

public class QpsController {

protected int qps = 10;

public QpsController(int qpsLimit){

this.qps = qpsLimit;

}

//实现control的具体逻辑 可以任意增加个人认为的必要函数

public void control(){

}

}

package com.aliyun.ezwork.test;

import java.util.concurrent.atomic.AtomicInteger;

/\*\*

\* Worker

\*

\*/

public class Worker {

static AtomicInteger counter = new AtomicInteger(0);

public void work(){

counter.incrementAndGet();

System.out.println(Thread.currentThread().getName() + " work ");

}

public static int getCount(){

return counter.get();

}

}

public class PrintDirectory{

public static void print(String path) {

//打印路径path 下的所有文件夹和文件名

}

}

import java.io.File;

public class PrintDirectory {

public static void print(String path, int tab) {

// 打印路径path下的所有文件夹和文件名

File file = new File(path);

File[] files = file.listFiles();

//子文件夹缩进一下

for (File a : files) {

for (int i = 0; i < tab; i++) {

System.out.print(" ");

}

//输出文件名

System.out.println(a.getName());

if (a.isDirectory()) {//如果当前是文件夹

print(a.getAbsolutePath(), tab + 1);//递归打印出文件夹中的东西，并缩进

}

}

}

public static void main(String[] args) {

print("E:/wechat/butterfly", 0);

}

}

public class ArrayInsertion{

private static List<Integer> sSorted = new LinkedList<>()

public static void addElement(int e) {

//请实现：该方法插入元素到sSorted，并保证sSorted永远保持从小到大排列；请勿使用List#sort等内置排序方法。

}

}

import java.util.Arrays;

import java.util.LinkedList;

import java.util.List;

public class ArrayInsertion {

private static List<Integer> sSorted = new LinkedList<>();

public static void adde(int e) {

LinkedList<Integer> tmpList = (LinkedList<Integer>) sSorted;

if (sSorted.size() == 0) {//如果列表里现在没数据，就添加到0

sSorted.add(e);

} else {

if (e < tmpList.getFirst()) {//比第一个元素小，就添加到第一个

tmpList.addFirst(e);

} else {

boolean insertFlag = false;

int index = 0;

for (Integer tmp : tmpList) {//否则遍历到第一个大的，就插入到前面

index++;

if (e < tmp) {

tmpList.add(index, tmp);

insertFlag = true;

break;

}

}

if (!insertFlag) {//如果没添加，就添加到最后一个

tmpList.addLast(e);

}

}

}

}

public static void main(String[] args) {

int[] arr = new int[]{3, 1, 3, 6, 9, 2, 7, 4, 9, 3, 11, 0, -1};

for (int i : arr) {

adde(i);

}

System.out.print(Arrays.toString(sSorted.toArray()));

}

}

### 3.3 GC

#### 1．（★）垃圾回收

##### 1.1概念

gc即垃圾收集机制是指jvm用于释放那些不再使用的对象所占用的内存。java语言并不要求jvm有gc，也没有规定gc如何工作。不过常用的jvm都有gc，而且大多数gc都使用类似的算法管理内存和执行收集操作。

　　在充分理解了垃圾收集算法和执行过程后，才能有效的优化它的性能。有些垃圾收集专用于特殊的应用程序。比如，实时应用程序主要是为了避免垃圾收集中断，而大多数OLTP应用程序则注重整体效率。理解了应用程序的工作负荷和jvm支持的垃圾收集算法，便可以进行优化配置垃圾收集器。

　　垃圾收集的目的在于清除不再使用的对象。gc通过确定对象是否被活动对象引用来确定是否收集该对象。gc首先要判断该对象是否是时候可以收集。两种常用的方法是引用计数和对象引用遍历。

##### 1.2.引用计数

　　引用计数存储对特定对象的所有引用数，也就是说，当应用程序创建引用以及引用超出范围时，jvm必须适当增减引用数。当某对象的引用数为0时，便可以进行垃圾收集。

##### 1.3.对象引用遍历

　　早期的jvm使用引用计数，现在大多数jvm采用对象引用遍历。对象引用遍历从一组对象开始，沿着整个对象图上的每条链接，递归确定可到达（reachable）的对象。如果某对象不能从这些根对象的一个（至少一个）到达，则将它作为垃圾收集。在对象遍历阶段，gc必须记住哪些对象可以到达，以便删除不可到达的对象，这称为标记（marking）对象。

　　下一步，gc要删除不可到达的对象。删除时，有些gc只是简单的扫描堆栈，删除未标记的未标记的对象，并释放它们的内存以生成新的对象，这叫做清除（sweeping）。这种方法的问题在于内存会分成好多小段，而它们不足以用于新的对象，但是组合起来却很大。因此，许多gc可以重新组织内存中的对象，并进行压缩（compact），形成可利用的空间。

　　为此，gc需要停止其他的活动活动。这种方法意味着所有与应用程序相关的工作停止，只有gc运行。结果，在响应期间增减了许多混杂请求。另外，更复杂的 gc不断增加或同时运行以减少或者清除应用程序的中断。有的gc使用单线程完成这项工作，有的则采用多线程以增加效率。

#### 2.（★★★）几种垃圾回收机制

##### 2.1.标记－清除收集器

　　这种收集器首先遍历对象图并标记可到达的对象，然后扫描堆栈以寻找未标记对象并释放它们的内存。这种收集器一般使用单线程工作并停止其他操作。

##### 2.2.标记－压缩收集器

　　有时也叫标记－清除－压缩收集器，与标记－清除收集器有相同的标记阶段。在第二阶段，则把标记对象复制到堆栈的新域中以便压缩堆栈。这种收集器也停止其他操作。

##### 2.3.复制收集器

　　这种收集器将堆栈分为两个域，常称为半空间。每次仅使用一半的空间，jvm生成的新对象则放在另一半空间中。gc运行时，它把可到达对象复制到另一半空间，从而压缩了堆栈。这种方法适用于短生存期的对象，持续复制长生存期的对象则导致效率降低。

##### 2.4.增量收集器

　　增量收集器把堆栈分为多个域，每次仅从一个域收集垃圾。这会造成较小的应用程序中断。

##### 2.5.分代收集器

　　这种收集器把堆栈分为两个或多个域，用以存放不同寿命的对象。jvm生成的新对象一般放在其中的某个域中。过一段时间，继续存在的对象将获得使用期并转入更长寿命的域中。分代收集器对不同的域使用不同的算法以优化性能。

##### 2.6.并发收集器

　　并发收集器与应用程序同时运行。这些收集器在某点上（比如压缩时）一般都不得不停止其他操作以完成特定的任务，但是因为其他应用程序可进行其他的后台操作，所以中断其他处理的实际时间大大降低。

##### 2.7.并行收集器

　　并行收集器使用某种传统的算法并使用多线程并行的执行它们的工作。在多cpu机器上使用多线程技术可以显著的提高java应用程序的可扩展性。

#### 3.（★★★）加速垃圾回收方式

##### 1.垃圾回收

对象是使用new创建的, 但是并没有与之相对应的delete操作来回收对象占用的内存. 当我们完成对某个对象的使用时, 只需停止该对象的引用:

->将引用改变为指向其他对象

->将引用指向null

->从方法中返回, 使得该方法的局部变量不复存在

要点:

->当我们从任何可执行代码都无法到达某个对象时, 它所占用的空间就可以被回收.

->垃圾回收意味着我们永远不用担心出现虚悬引用(dangling reference). 虚悬引用, 指得是引用已经被删除的内存空间. 在那些程序员可以直接控制何时删除对象的系统中, 会存在这样的问题.

->垃圾回收器模型: 引用计数器法(不能解决循环引用), 标记-清除(mark-and-sweep)

##### 2.终结

finalize方法

->在垃圾回收器确定该对象是不可达的且该对象的空间将被回收之后, 垃圾回收器就会调用这个方法.

->这个方法可以清除该对象所使用的所有非内存资源, 对每一个对象最多只能调用一次, 即使在这个方法的执行使得该对象重新变为可达之后又马上会再次变为不可达的情况下, 该方法也只能调用一次.

->finalize方法可以在任何特定的时间段内被调用, 它也可能永远不会被调用(java虚拟机结束).

覆写finalize方法

->当一个对象变成垃圾时, 它所引用的其他对象也很有可能会变成垃圾. 这些垃圾可能在调用我们编写的finalize方法之前就已经被终结了, 因此它们可能处于不可预知的状态.

->覆写finalize方法是, 加上super.finalize方法. 最好加在finally字句里面.保证其超类中声明的部分内容也可以被终结.

##### 3.与垃圾回收器交互的相关类和方法

类: Runtime.getRuntime(), System

方法:gc(), runFinalization(), freeMemory(), totalMemory(), maxMemory()

System类支持静态的gc()和runFinalization()方法, 它们将调用当前Runtime对象上的相应方法.

##### 4.可达性状态和引用对象

对象只有在没有任何引用指定它的时候才可以被当作垃圾回收, 但有时我们可能希望在仍旧有选定引用指向对象时, 将该对象作为垃圾回收掉。

引用对象的唯一用途就是维护对另一个被称为指称物(referent)的对象的引用. 通常我们通过字段或者局部变量来维护对对象的引用, 但是现在我们可以维护对引用对象的直接引用, 而该引用对象包装了我们实际需要的对象. 垃圾回收器可能判断出对某个对象的残留引用是否都是经由引用对象面引用到该对象的, 因此它可以决定是否要回收该对象. 引用对象的强度将决定垃圾回收器的行为, 普通的引用都是强度最大的引用.

Reference类

->包:java.lang.ref

->典型方法: get(), clear(), enqueue(), isEnqueued()

引用和可达性强度

->对象是强可达的(strongly reachable):普通的引用

->对象是软可达的(softly reachable):SoftReference

->对象是弱可达的(weakly reachable):WeakReference

->对象是虚可达的(phantom reachable):PhantomReference

->对象是不可达的:没有引用链接

一旦对象变为弱可达的(或者列弱), 它就可以被终结. 如果在终结之后该对象是不可达的那么它就可以被回收了.

对象可达性阶段会触发垃圾回收器对相关的引用对象类型做出适当的行为:

->软可达对象可能会任凭垃圾回收器去回收. 我们可确定的是所有对软可达对象的SoftReference都会在抛出outofMemoryError错误这前被清除.

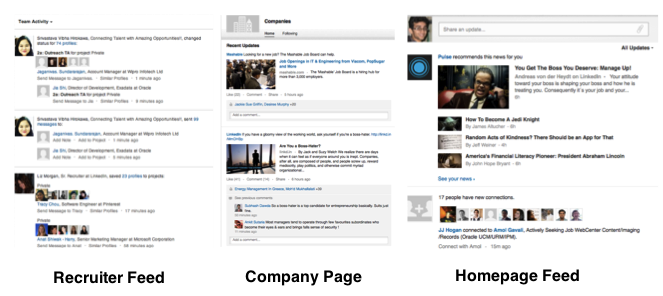
->弱可达对象将会被垃圾回收器回收.

->虚可达对象并不是真正意义上的可达, 因为无法通过PhantomReference访问其指称对象, 其get方法总是返回null. 但是虚引用的存在可以防止对象在显式清除虚引用之前被回收. 虚引用使我们可以处理那些finalize方法已经被调用过的对象, 从而可以安全地认为它们是"死"的。

#### 4. （★★★）高吞吐低延迟Java应用的垃圾回收优化

高性能应用构成了现代网络的支柱。LinkedIn有许多内部高吞吐量服务来满足每秒数千次的用户请求。要优化用户体验，低延迟地响应这些请求非常重要。

比如说，用户经常用到的一个功能是了解动态信息——不断更新的专业活动和内容的列表。动态信息在LinkedIn随处可见，包括公司页面，学校页面以及最重要的主页。基础动态信息数据平台为我们的经济图谱(会员，公司，群组等等)中各种实体的更新建立索引，它必须高吞吐低延迟地实现相关的更新。

[](http://www.importnew.com/?attachment_id=11339)

*图1 LinkedIn 动态信息*

这些高吞吐低延迟的Java应用转变为产品，开发人员必须确保应用开发周期的每个阶段一致的性能。确定优化垃圾回收(Garbage Collection,GC)的设置对达到这些指标非常关键。

本文章通过一系列步骤来明确需求并优化GC，目标读者是为实现应用的高吞吐低延迟，对使用系统方法优化GC感兴趣的开发人员。文章中的方法来自于LinkedIn构建下一代动态信息数据平台过程。这些方法包括但不局限于以下几点：[并发标记清除](http://www.oracle.com/technetwork/java/javase/memorymanagement-whitepaper-150215.pdf)(Concurrent Mark Sweep,CMS)和[G1](http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/G1GettingStarted/index.html)垃圾回收器的CPU和内存开销，避免长期存活对象引起的持续GC周期，优化GC线程任务分配使性能提升，以及GC停顿时间可预测所需的OS设置。

#### 5. （★★★）优化GC的正确时机？

GC运行随着代码级的优化和工作负载而发生变化。因此在一个已实施性能优化的接近完成的代码库上调整GC非常重要。但是在端到端的基本原型上进行初步分析也很有必要，该原型系统使用存根代码并模拟了可代表产品环境的工作负载。这样可以捕捉该架构延迟和吞吐量的真实边界，进而决定是否纵向或横向扩展。

在下一代动态信息数据平台的原型阶段，几乎实现了所有端到端的功能，并且模拟了当前产品基础架构所服务的查询负载。从中我们获得了多种用来衡量应用性能的工作负载特征和足够长时间运行情况下的GC特征。

#### 6. （★★★）优化GC的步骤

下面是为满足高吞吐，低延迟需求优化GC的总体步骤。也包括在动态信息数据平台原型实施的具体细节。可以看到在ParNew/CMS有最好的性能，但我们也实验了G1垃圾回收器。

##### 1.理解GC基础知识

理解GC工作机制非常重要，因为需要调整大量的参数。Oracle的Hotspot JVM 内存管理[白皮书](http://www.oracle.com/technetwork/java/javase/memorymanagement-whitepaper-150215.pdf)是开始学习Hotspot JVM GC算法非常好的资料。了解G1垃圾回收器，请查看该[论文](http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/G1GettingStarted/index.html)。

##### 2. 仔细考量GC需求

为降低应用性能的GC开销，可以优化GC的一些特征。吞吐量、延迟等这些GC特征应该长时间测试运行观察，确保特征数据来自于应用程序的处理对象数量发生变化的多个GC周期。

* Stop-the-world回收器回收垃圾时会暂停应用线程。停顿的时长和频率不应该对应用遵守SLA产生不利的影响。
* 并发GC算法与应用线程竞争CPU周期。这个开销不应该影响应用吞吐量。
* 不压缩GC算法会引起堆碎片化，导致full GC长时间Stop-the-world停顿。
* 垃圾回收工作需要占用内存。一些GC算法产生更高的内存占用。如果应用程序需要较大的堆空间，要确保GC的内存开销不能太大。
* 清晰地了解GC日志和常用的JVM参数对简单调整GC运行很有必要。GC运行随着代码复杂度增长或者工作特性变化而改变。

我们使用Linux OS的Hotspot [Java7u51](http://www.oracle.com/technetwork/java/javase/7u51-relnotes-2085002.html)，32GB堆内存，6GB新生代(young generation)和-XX:CMSInitiatingOccupancyFraction值为70(老年代GC触发时其空间占用率)开始实验。设置较大的堆内存用来维持长期存活对象的对象缓存。一旦这个缓存被填充，提升到老年代的对象比例显著下降。

使用初始的GC配置，每三秒发生一次80ms的新生代GC停顿，超过百分之99.9的应用延迟100ms。这样的GC很可能适合于SLA不太严格要求延迟的许多应用。然而，我们的目标是尽可能降低百分之99.9应用的延迟，为此GC优化是必不可少的。

##### 3.理解GC指标

优化之前要先衡量。了解GC日志的详细细节(使用这些选项：-XX:+PrintGCDetails -XX:+PrintGCTimeStamps -XX:+PrintGCDateStamps -XX:+PrintTenuringDistribution -XX:+PrintGCApplicationStoppedTime)可以对该应用的GC特征有总体的把握。

LinkedIn的内部监控和报表系统，[inGraphs](https://engineering.linkedin.com/32/eric-intern-origin-ingraphs)和[Naarad](https://github.com/linkedin/naarad)，生成了各种有用的指标可视化图形，比如GC停顿时间百分比，一次停顿最大持续时间，长时间内GC频率。除了Naarad，有很多开源工具比如[gclogviewer](https://code.google.com/p/gclogviewer/)可以从GC日志创建可视化图形。

在这个阶段，需要确定GC频率和停顿时长是否影响应用满足延迟性需求的能力。

##### 4.降低GC频率

在分代GC算法中，降低回收频率可以通过：(1)降低对象分配/提升率；(2)增加代空间的大小。

在Hotspot JVM中，新生代GC停顿时间取决于一次垃圾回收后对象的数量，而不是新生代自身的大小。增加新生代大小对于应用性能的影响需要仔细评估：

* 如果更多的数据存活而且被复制到survivor区域，或者每次垃圾回收更多的数据提升到老年代，增加新生代大小可能导致更长的新生代GC停顿。
* 另一方面，如果每次垃圾回收后存活对象数量不会大幅增加，停顿时间可能不会延长。在这种情况下，减少GC频率可能使应用总体延迟降低和(或)吞吐量增加。

对于大部分为短期存活对象的应用，仅仅需要控制前面所说的参数。对于创建长期存活对象的应用，就需要注意，被提升的对象可能很长时间都不能被老年代GC周期回收。如果老年代GC触发阈值(老年代空间占用率百分比)比较低，应用将陷入不断的GC周期。设置高的GC触发阈值可避免这一问题。

由于我们的应用在堆中维持了长期存活对象的较大缓存，将老年代GC触发阈值设置为-XX:CMSInitiatingOccupancyFraction=92 -XX:+UseCMSInitiatingOccupancyOnly。我们也试图增加新生代大小来减少新生代回收频率，但是并没有采用，因为这增加了应用延迟。

##### 5.缩短GC停顿时间

减少新生代大小可以缩短新生代GC停顿时间，因为这样被复制到survivor区域或者被提升的数据更少。但是，正如前面提到的，我们要观察减少新生代大小和由此导致的GC频率增加对于整体应用吞吐量和延迟的影响。新生代GC停顿时间也依赖于tenuring threshold(提升阈值)和空间大小(见第6步)。

使用CMS尝试最小化堆碎片和与之关联的老年代垃圾回收full GC停顿时间。通过控制对象提升比例和减小-XX:CMSInitiatingOccupancyFraction的值使老年代GC在低阈值时触发。所有选项的细节调整和他们相关的权衡，请查看[Web Services的Java 垃圾回收](http://engineering.linkedin.com/26/tuning-java-garbage-collection-web-services)和[Java 垃圾回收精粹](http://mechanical-sympathy.blogspot.com/2013/07/java-garbage-collection-distilled.html)。

我们观察到Eden区域的大部分新生代被回收，几乎没有对象在survivor区域死亡，所以我们将tenuring threshold从8降低到2(使用选项：-XX:MaxTenuringThreshold=2),为的是缩短新生代垃圾回收消耗在数据复制上的时间。

我们也注意到新生代回收停顿时间随着老年代空间占用率上升而延长。这意味着来自老年代的压力使得对象提升花费更多的时间。为解决这个问题，将总的堆内存大小增加到40GB，减小-XX:CMSInitiatingOccupancyFraction的值到80，更快地开始老年代回收。尽管-XX:CMSInitiatingOccupancyFraction的值减小了，增大堆内存可以避免不断的老年代GC。在本阶段，我们获得了70ms新生代回收停顿和百分之99.9延迟80ms。

##### 6.优化GC工作线程的任务分配

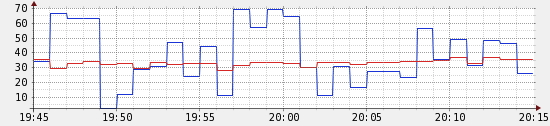
进一步缩短新生代停顿时间，我们决定研究优化与GC线程绑定任务的选项。

-XX:ParGCCardsPerStrideChunk 选项控制GC工作线程的任务粒度，可以帮助不使用[补丁](http://www.importnew.com/%5bhttp:/bugs.java.com/bugdatabase/view_bug.do?bug_id=7068625%5d(http://bugs.java.com/bugdatabase/view_bug.do?bug_id=7068625))而获得最佳性能，这个补丁用来优化新生代垃圾回收的[卡表扫描时间](http://blog.ragozin.info/2012/03/secret-hotspot-option-improving-gc.html)。有趣的是新生代GC时间随着老年代空间的增加而延长。将这个选项值设为32678，新生代回收停顿时间降低到平均50ms。此时百分之99.9应用延迟60ms。

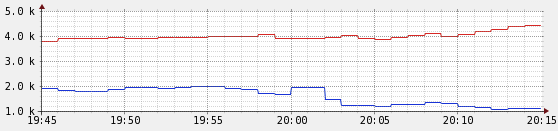
也有其他选项将任务映射到GC线程，如果OS允许的话，-XX:+BindGCTaskThreadsToCPUs选项绑定GC线程到个别的CPU核。-XX:+UseGCTaskAffinity使用affinity参数将任务分配给GC工作线程。然而，我们的应用并没有从这些选项发现任何益处。实际上，一些调查显示这些选项在Linux系统不起作用[1,2]。

##### 7.了解GC的CPU和内存开销

并发GC通常会增加CPU的使用。我们观察了运行良好的CMS默认设置，并发GC和G1垃圾回收器共同工作引起的CPU使用增加显著降低了应用的吞吐量和延迟。与CMS相比，G1可能占用了应用更多的内存开销。对于低吞吐量的非计算密集型应用，GC的高CPU使用率可能不需要担心。

[](http://www.importnew.com/?attachment_id=11340)

*图2 ParNew/CMS和G1的CPU使用百分数%：相对来说CPU使用率变化明显的节点使用G1  
选项-XX:G1RSetUpdatingPauseTimePercent=20*

[](http://www.importnew.com/?attachment_id=11338)

*图3 ParNew/CMS和G1每秒服务的请求数：吞吐量较低的节点使用G1  
选项-XX:G1RSetUpdatingPauseTimePercent=20*

##### 8.为GC优化系统内存和I/O管理

通常来说，GC停顿发生在(1)低用户时间，高系统时间和高时钟时间和(2)低用户时间，低系统时间和高时钟时间。这意味着基础的进程/OS设置存在问题。情况(1)可能说明Linux从JVM偷页，情况(2)可能说明清除磁盘缓存时Linux启动GC线程，等待I/O时线程陷入内核。在这些情况下如何设置参数可以参考[该PPT](http://www.slideshare.net/cuonghuutran/gc-andpagescanattacksbylinux)。

为避免运行时性能损失，启动应用时使用JVM选项-XX:+AlwaysPreTouch访问和清零页面。设置vm.swappiness为零，除非在绝对必要时，OS不会交换页面。

可能你会使用mlock将JVM页pin在内存中，使OS不换出页面。但是，如果系统用尽了所有的内存和交换空间，OS通过kill进程来回收内存。通常情况下，Linux内核会选择高驻留内存占用但还没有长时间运行的进程([OOM情况下killing进程的工作流](https://www.kernel.org/doc/gorman/html/understand/understand016.html))。对我们而言，这个进程很有可能就是我们的应用程序。一个服务具备优雅降级(适度退化)的特点会更好，服务突然故障预示着不太好的可操作性——因此，我们没有使用mlock而是vm.swappiness避免可能的交换惩罚。

#### 7.GC习题

##### **1.(**★**)**GC是什么？为什么要有GC？

答：GC是垃圾回收机制

原因：内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java提供的GC功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java语言没有提供释放已分配内存的显示操作方法。

##### 2.(★)如下代码

10 public Object m(){

11 Object o = new Float(3.14F);

12 Object[] oa = new Object[1];

13 oa[0] = o;

14 o = null;

15 print “return 0”;B

16 }

当Float 对象在第11行被创建后，什么时候能够被垃圾回收？B

1. 13行以后 B. 14行以后 C.15行以后 D. 不会回收

### 3.4 JVM

#### 1.（★★★） 谈谈对jvm的了解

帮助我们实现跨平台的虚拟机器，提供了很多与平台相关的解释器，将class文件通过解释器解释成与平台相关的机器码，另外，它能够解析内存和计时器，调度CPU，完成java的垃圾回收。

JVM是java的核心和基础，在java编译器和os平台之间的虚拟处理器。它是一种利用软件方法实现的抽象的计算机基于下层的操作系统和硬件平台，可以在上面执行java的字节码程序。

java编译器只要面向JVM，生成JVM能理解的代码或字节码文件。Java源文件经编译成字节码程序，通过JVM将每一条指令翻译成不同平台机器码，通过特定平台运行。

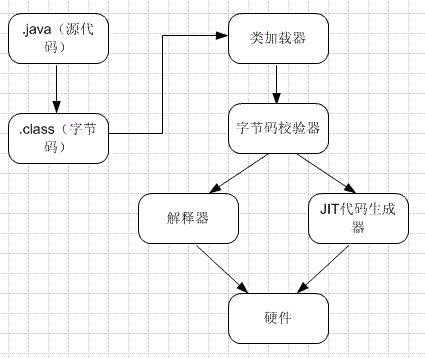
JVM执行程序的过程 ：

I.加载.class文件

II.管理并分配内存

III.执行垃圾收集

JRE（java运行时环境）由JVM构造的java程序的运行环境



参考: <https://blog.csdn.net/g19920917/article/details/40015611>

更多参考：https://blog.csdn.net/qq\_38019655/article/details/82289013

#### 2.（★★★） Jvm调优了解哪些？

答：

垃圾回收算法

按照基本回收策略分为引用计数、标记-清除、复制、标记-整理

按分区对待的方式分为增量收集、分代收集

按系统线程分串行收集、并行收集和并发收集

**如何处理碎片**

由于不同Java对象存活时间是不一定的，因此，在程序运行一段时间以后，如果不进行内存整理，就会出现零散的内存碎片。碎片最直接的问题就是会导致无法分配大块的内存空间，以及程序运行效率降低。所以，在上面提到的基本垃圾回收算法中，“复制”方式和“标记-整理”方式，都可以解决碎片的问题。

**如何解决同时存在的对象创建和对象回收问题**

垃圾回收线程是回收内存的，而程序运行线程则是消耗（或分配）内存的，一个回收内存，一个分配内存，从这点看，两者是矛盾的。因此，在现有的垃圾回收方式中，要进行垃圾回收前，一般都需要暂停整个应用（即：暂停内存的分配），然后进行垃圾回收，回收完成后再继续应用。这种实现方式是最直接，而且最有效的解决二者矛盾的方式。

但是这种方式有一个很明显的弊端，就是当堆空间持续增大时，垃圾回收的时间也将会相应的持续增大，对应应用暂停的时间也会相应的增大。一些对相应时间要求很高的应用，比如最大暂停时间要求是几百毫秒，那么当堆空间大于几个G时，就很有可能超过这个限制，在这种情况下，垃圾回收将会成为系统运行的一个瓶颈。为解决这种矛盾，有了并发垃圾回收算法，使用这种算法，垃圾回收线程与程序运行线程同时运行。在这种方式下，解决了暂停的问题，但是因为需要在新生成对象的同时又要回收对象，算法复杂性会大大增加，系统的处理能力也会相应降低，同时，“碎片”问题将会比较难解决。

更多优化策略参考: <https://pengjiaheng.iteye.com/blog/518623>

11章JVM调优

#### 3. (★★★)JVM内存中存储内存和执行内存保存哪些数据，如果有堆外内存，存储内存还有嘛？

运行时数据区通常包括这几个部分：程序计数器(Program Counter Register)、Java栈(VM Stack)、本地方法栈(Native Method Stack)、方法区(Method Area)、堆(Heap)。

各个数据区的含义参见<https://www.cnblogs.com/dolphin0520/p/3613043.html>

堆外内存概念详见<https://www.jianshu.com/p/50be08b54bee>

如果有堆外内存，存储内存还有。

#### 4. （★★★）JVM堆大小的调整

　　Sun HotSpot 1.4.1使用分代收集器，它把堆分为三个主要的域：新域、旧域以及永久域。Jvm生成的所有新对象放在新域中。一旦对象经历了一定数量的垃圾收集循环后，便获得使用期并进入旧域。在永久域中jvm则存储class和method对象。就配置而言，永久域是一个独立域并且不认为是堆的一部分。

　　下面介绍如何控制这些域的大小。可使用-Xms和-Xmx 控制整个堆的原始大小或最大值。

　　下面的命令是把初始大小设置为128M：

　　java –Xms128m

　　–Xmx256m为控制新域的大小，可使用-XX:NewRatio设置新域在堆中所占的比例。

　　下面的命令把整个堆设置成128m，新域比率设置成3，即新域与旧域比例为1：3，新域为堆的1/4或32M：

java –Xms128m –Xmx128m

–XX:NewRatio =3可使用-XX:NewSize和-XX:MaxNewsize设置新域的初始值和最大值。

　　下面的命令把新域的初始值和最大值设置成64m:

java –Xms256m –Xmx256m –Xmn64m

　　永久域默认大小为4m。运行程序时，jvm会调整永久域的大小以满足需要。每次调整时，jvm会对堆进行一次完全的垃圾收集。

　　使用-XX:MaxPerSize标志来增加永久域搭大小。在WebLogic Server应用程序加载较多类时，经常需要增加永久域的最大值。当jvm加载类时，永久域中的对象急剧增加，从而使jvm不断调整永久域大小。为了避免调整，可使用-XX:PerSize标志设置初始值。

　　下面把永久域初始值设置成32m，最大值设置成64m。

java -Xms512m -Xmx512m -Xmn128m -XX:PermSize=32m -XX:MaxPermSize=64m

　　默认状态下，HotSpot在新域中使用复制收集器。该域一般分为三个部分。第一部分为Eden，用于生成新的对象。另两部分称为救助空间，当Eden 充满时，收集器停止应用程序，把所有可到达对象复制到当前的from救助空间，一旦当前的from救助空间充满，收集器则把可到达对象复制到当前的to救助空间。From和to救助空间互换角色。维持活动的对象将在救助空间不断复制，直到它们获得使用期并转入旧域。使用-XX:SurvivorRatio 可控制新域子空间的大小。

　　同NewRation一样，SurvivorRation规定某救助域与Eden空间的比值。比如，以下命令把新域设置成64m，Eden占32m，每个救助域各占16m：

java -Xms256m -Xmx256m -Xmn64m -XX:SurvivorRation =2

　　如前所述，默认状态下HotSpot对新域使用复制收集器，对旧域使用标记－清除－压缩收集器。在新域中使用复制收集器有很多意义，因为应用程序生成的大部分对象是短寿命的。理想状态下，所有过渡对象在移出Eden空间时将被收集。如果能够这样的话，并且移出Eden空间的对象是长寿命的，那么理论上可以立即把它们移进旧域，避免在救助空间反复复制。但是，应用程序不能适合这种理想状态，因为它们有一小部分中长寿命的对象。最好是保持这些中长寿命的对象并放在新域中，因为复制小部分的对象总比压缩旧域廉价。为控制新域中对象的复制，可用-XX:TargetSurvivorRatio控制救助空间的比例（该值是设置救助空间的使用比例。如救助空间位1M，该值50表示可用500K）。该值是一个百分比，默认值是50。当较大的堆栈使用较低的 sruvivorratio时，应增加该值到80至90，以更好利用救助空间。用-XX:maxtenuring threshold可控制上限。

　　为放置所有的复制全部发生以及希望对象从eden扩展到旧域，可以把MaxTenuring Threshold设置成0。设置完成后，实际上就不再使用救助空间了，因此应把SurvivorRatio设成最大值以最大化Eden空间，设置如下：

java … -XX:MaxTenuringThreshold=0 –XX:SurvivorRatio＝50000 …

#### 5. （★★★）BEA JRockit JVM的使用

　　Bea WebLogic 8.1使用的新的JVM用于Intel平台。在Bea安装完毕的目录下可以看到有一个类似于jrockit81sp1\_141\_03的文件夹。这就是 Bea新JVM所在目录。不同于HotSpot把Java字节码编译成本地码，它预先编译成类。JRockit还提供了更细致的功能用以观察JVM的运行状态，主要是独立的GUI控制台（只能适用于使用Jrockit才能使用jrockit81sp1\_141\_03自带的console监控一些cpu及 memory参数）或者WebLogic Server控制台。

　　Bea JRockit JVM支持4种垃圾收集器：

##### 1.分代复制收集器

　　它与默认的分代收集器工作策略类似。对象在新域中分配，即JRockit文档中的nursery。这种收集器最适合单cpu机上小型堆操作。

##### 2.单空间并发收集器

　　该收集器使用完整堆，并与背景线程共同工作。尽管这种收集器可以消除中断，但是收集器需花费较长的时间寻找死对象，而且处理应用程序时收集器经常运行。如果处理器不能应付应用程序产生的垃圾，它会中断应用程序并关闭收集。

　　分代并发收集器这种收集器在护理域使用排它复制收集器，在旧域中则使用并发收集器。由于它比单空间共同发生收集器中断频繁，因此它需要较少的内存，应用程序的运行效率也较高，注意，过小的护理域可以导致大量的临时对象被扩展到旧域中。这会造成收集器超负荷运作，甚至采用排它性工作方式完成收集。

##### 3.并行收集器

　　该收集器也停止其他进程的工作，但使用多线程以加速收集进程。尽管它比其他的收集器易于引起长时间的中断，但一般能更好的利用内存，程序效率也较高。

　　默认状态下，JRockit使用分代并发收集器。要改变收集器，可使用-Xgc:，对应四个收集器分别为 gencopy，singlecon，gencon以及parallel。可使用-Xms和-Xmx设置堆的初始大小和最大值。要设置护理域，则使用- Xns:java –jrockit –Xms512m –Xmx512m –Xgc:gencon –Xns128m…尽管JRockit支持-verbose:gc开关，但它输出的信息会因收集器的不同而异。JRockit还支持memory、 load和codegen的输出。

　　注意 ：如果 使用JRockit JVM的话还可以使用WLS自带的console（C:\bea\jrockit81sp1\_141\_03\bin下）来监控一些数据，如cpu， memery等。要想能构监控必须在启动服务时startWeblogic.cmd中加入－Xmanagement参数。

#### 6. （★★★）如何从JVM中获取信息来进行调整

　　-verbose.gc开关可显示gc的操作内容。打开它，可以显示最忙和最空闲收集行为发生的时间、收集前后的内存大小、收集需要的时间等。打开- xx:+ printgcdetails开关，可以详细了解gc中的变化。打开-XX: + PrintGCTimeStamps开关，可以了解这些垃圾收集发生的时间，自jvm启动以后以秒计量。最后，通过-xx: + PrintHeapAtGC开关了解堆的更详细的信息。为了了解新域的情况，可以通过-XX:=PrintTenuringDistribution开关了解获得使用期的对象权。

#### 7. （★★★）Pdm系统JVM调整

##### 1.服务器：前提内存1G 单CPU

　　可通过如下参数进行调整：－server 启用服务器模式（如果CPU多，服务器机建议使用此项）

　　－Xms,－Xmx一般设为同样大小。 800m

　　－Xmn 是将NewSize与MaxNewSize设为一致。320m

　　－XX:PerSize 64m

　　－XX:NewSize 320m 此值设大可调大新对象区，减少Full GC次数

　　－XX:MaxNewSize 320m

　　－XX:NewRato NewSize设了可不设。

　　－XX: SurvivorRatio

　　－XX:userParNewGC 可用来设置并行收集

　　－XX:ParallelGCThreads 可用来增加并行度

　　－XXUseParallelGC 设置后可以使用并行清除收集器

　　－XX：UseAdaptiveSizePolicy 与上面一个联合使用效果更好，利用它可以自动优化新域大小以及救助空间比值

##### 2.客户机：通过在JNLP文件中设置参数来调整客户端JVM

　　JNLP中参数：initial-heap-size和max-heap-size

　　这可以在framework的RequestManager中生成JNLP文件时加入上述参数，但是这些值是要求根据客户机的硬件状态变化的（如客户机的内存大小等）。建议这两个参数值设为客户机可用内存的60％（有待测试）。为了在动态生成JNLP时以上两个参数值能够随客户机不同而不同，可靠虑获得客户机系统信息并将这些嵌到首页index.jsp中作为连接请求的参数。

　　在设置了上述参数后可以通过Visualgc 来观察垃圾回收的一些参数状态，再做相应的调整来改善性能。一般的标准是减少fullgc的次数，最好硬件支持使用并行垃圾回收（要求多CPU）。

#### 8. （★★★）类装载器

顾名思义，就是用来装载.class文件的。JVM的两种类装载器包括：启动类装载器和用户自定义类装载器，启动类装载器是JVM实现的一部分，用户自定义类装载器则是Java程序的一部分，必须是ClassLoader类的子类。（下面所述情况是针对Sun JDK1.2）

动类装载器：只在系统类(java API的类文件)的安装路径查找要装入的类

用户自定义类装载器：

系统类装载器：在JVM启动时创建，用来在CLASSPATH目录下查找要装入的类

其他用户自定义类装载器：这里有必要先说一下ClassLoader类的几个方法，了解它们对于了解自定义类装载器如何装载.class文件至关重要。

protected final Class defineClass(String name, byte data[], int offset, int length)

protected final Class defineClass(String name, byte data[], int offset, int length, ProtectionDomain protectionDomain);

protected final Class findSystemClass(String name)

protected final void resolveClass(Class c)

defineClass用来将二进制class文件（新类型）导入到方法区,也就是这里指的类是用户自定义的类（也就是负责装载类）

findSystemClass通过类型的全限定名，先通过系统类装载器或者启动类装载器来装载，并返回Class对象。

ResolveClass: 让类装载器进行连接动作（包括验证，分配内存初始化，将类型中的符号引用解析为直接引用），这里涉及到java命名空间的问题，JVM保证被一个类装载器装载的类所引用的所有类都被这个类装载器装载，同一个类装载器装载的类之间可以相互访问，但是不同类装载器装载的类看不见对方，从而实现了有效的屏蔽。

#### 9. （★★★）执行引擎：它或者在执行字节码，或者执行本地方法

   要说执行引擎，就不得不的指令集，每一条指令包含一个单字节的操作码，后面跟0个或者多个操作数。

（一）指令集以栈为设计中心，而非以寄存器为中心

这种指令集设计如何满足Java体系的要求：

平台无关性：以栈为中心使得在只有很少register的机器上实现java更便利

compiler一般采用stack向连接优化器传递编译的中间结果，若指令集以stack为基础，则有利于运行时进行的优化工作与执行即时编译或者自适应优化的执行引擎结合，通俗的说就是使编译和运行用的数据结构统一，更有利于优化的开展。

网络移动性：class文件的紧凑性。

安全性：指令集中绝大部分操作码都指明了操作的类型。（在装载的时候使用数据流分析期进行一次性验证，而非在执行每条指令的时候进行验证，有利于提高执行速度）。

1. 执行技术

主要的执行技术有:解释，即时编译，自适应优化、芯片级直接执行

其中解释属于第一代JVM，即时编译JIT属于第二代JVM，自适应优化（目前Sun的HotspotJVM采用这种技术）则吸取第一代JVM和第二代JVM的经验，采用两者结合的方式

自适应优化：开始对所有的代码都采取解释执行的方式，并监视代码执行情况，然后对那些经常调用的方法启动一个后台线程，将其编译为本地代码，并进行仔细优化。若方法不再频繁使用，则取消编译过的代码，仍对其进行解释执行。

运行时数据区：主要包括：方法区，堆，java栈，PC寄存器，本地方法栈

1. 方法区和堆由所有线程共享   
    堆：存放所有程序在运行时创建的对象  
    方法区：当JVM的类装载器加载.class文件，并进行解析，把解析的类型信息放入方法区。
2. Java栈和PC寄存器由线程独享，在新线程创建时间里
3. 本地方法栈： 存储本地方法调用的状态

上边总体介绍了运行时数据区的主要内容，下边进行详细介绍，要介绍数据区，就不得不说明JVM中的数据类型。

JVM中的数据类型：JVM中基本的数据单元是word,而word的长度由JVM具体的实现者来决定

数据类型包括基本类型和引用类型，

1. 基本类型

包括：数值类型(包括除boolean外的所有的java基本数据类型)，boolean（在JVM中使用int来表示，0表示false，其他int值均表示true）和returnAddress（JVM的内部类型，用来实现finally子句）。

1. 引用类型包括：数组类型，类类型，接口类型

前边讲述了JVM中数据的表示，下面让我们输入到JVM的数据区

首先来看方法区：

上边已经提到，方法区主要用来存储JVM从class文件中提取的类型信息，那么类型信息是如何存储的呢？众所周知，Java使用的是大端序（big—endian:即低字节的数据存储在高位内存上，如对于1234，12是高位数据，34为低位数据，则java中的存储格式应该为12存在内存的低地址，34存在内存的高地址，x86中的存储格式与之相反）来存储数据，这实际上是在class文件中数据的存储格式，但是当数据倒入到方法区中时，JVM可以以任何方式来存储它。  
类型信息：包括class的全限定名，class的直接父类，类类型还是接口类型，类的修饰符（public,等），所有直接父接口的列表，Class对象提供了访问这些信息的窗口（可通过Class.forName(“”)或instance.getClass()获得），下面是Class的方法，相信大家看了会恍然大悟，（原来如此J）

getName(), getSuperClass(), isInterface(), getInterfaces(), getClassLoader();

static变量作为类型信息的一部分保存

指向ClassLoader类的引用：在动态连接时装载该类中引用的其他类

指向Class类的引用：必然的，上边已述

该类型的常量池：包括直接常量（String，integer和float point常量）以及对其他类型、字段和方法的符号引用（注意：这里的常量池并不是普通意义上的存储常量的地方，这些符号引用可能是我们在编程中所接触到的变量），由于这些符号引用，使得常量池成为java程序动态连接中至关重要的部分

字段信息：普通意义上的类型中声明的字段

方法信息：类型中各个方法的信息

编译期常量：指用final声明或者用编译时已知的值初始化的类变量

class将所有的常量复制至其常量池或者其字节码流中。

方法表：一个数组，包括所有它的实例可能调用的实例方法的直接引用（包括从父类中继承来的）

除此之外，若某个类不是抽象和本地的，还要保存方法的字节码，操作数栈和该方法的栈帧，异常表。

举例：

class Lava{  
  private int speed = 5;  
  void flow(){}  
}  
class Volcano{  
  public static void main(String[] args){  
 Lava lava = new Lava();  
 lava.flow();

}

}  
运行命令java Volcano;  
（1）JVM找到Volcano.class倒入，并提取相应的类型信息到方法区。通过执行方法区中的字节码，JVM执行main（）方法，（执行时会一直保存指向Vocano类的常量池的指针）

1. Main（）中第一条指令告诉JVM需为列在常量池第一项的类分配内存（此处再次说明了常量池并非只存储常量信息），然后JVM找到常量池的第一项，发现是对Lava类的符号引用，则检查方法区，看Lava类是否装载，结果是还未装载，则查找“Lava.class”,将类型信息写入方法区，并将方法区Lava类信息的指针来替换Volcano原常量池中的符号引用，即用直接引用来替换符号引用。

（3 )JVM看到new关键字，准备为Lava分配内存，根据Volcano的常量池的第一项找到Lava在方法区的位置，并分析需要多少对空间，确定后，在堆上分配空间，并将speed变量初始为0，并将lava对象的引用压到栈中

（4调用lava的flow（）方法

好了，大致了解了方法区的内容后，让我们来看看堆

java对象的堆实现：

java对象主要由实例变量（包括自己所属的类和其父类声明的）以及指向方法区中类数据的指针，指向方法表的指针，对象锁（非必需）， 等待集合（非必需），GC相关的数据（非必需）（主要视GC算法而定，如对于标记并清除算法，需要标记对象是否被引用，以及是否已调用finalize（）方法）。

那么为什么java对象中要有指向类数据的指针呢？我们从几个方面来考虑

首先：当程序中将一个对象引用转为另一个类型时，如何检查转换是否允许？需用到类数据

其次：动态绑定时，并不是需要引用类型，而是需要运行时类型  
这里的迷惑是：为什么类数据中保存的是实际类型，而非引用类型？这个问题先留下来，我想在后续的读书笔记中应该能明白

指向方法表的指针：这里和C++的VTBL是类似的，有利于提高方法调用的效率

对象锁：用来实现多个线程对共享数据的互斥访问

等待集合：用来让多个线程为完成共同目标而协调功过。（注意Object类中的wait(),notify(),notifyAll()方法）。

Java数组的堆实现：数组也拥有一个和他们的类相关联的Class实例，具有相同dimension和type的数组是同一个类的实例。数组类名的表示：如[[Ljava/lang/Object 表示Object[][]，[I表示int[]，[[[B表示byte[][][]

至此，堆已大致介绍完毕，下面来介绍程序计数器和java栈

程序计数器：为每个线程独有，在线程启动时创建，

  若thread执行java方法，则PC保存下一条执行指令的地址。

  若thread执行native方法，则Pc的值为undefined

Java栈：java栈以帧为单位保存线程的运行状态，java栈只有两种操作，帧的压栈和出栈。

每个帧代表一个方法，java方法有两种返回方式，return和抛出异常，两种方式都会导致该方法对应的帧出栈和释放内存。

帧的组成：局部变量区（包括方法参数和局部变量，对于instance方法，还要首先保存this类型，其中方法参数按照声明顺序严格放置，局部变量可以任意放置），操作数栈，帧数据区（用来帮助支持常量池的解析，正常方法返回和异常处理）。

本地方法栈：依赖于本地方法的实现，如某个JVM实现的本地方法借口使用C连接模型，则本地方法栈就是C栈，可以说某线程在调用本地方法时，就进入了一个不受JVM限制的领域，也就是JVM可以利用本地方法来动态扩展本身。

#### 10.(★★★)JVM的内存如何分配？如何回收？

1、当JVM给一个对象分配内存的时候，如果启动了本地线程分配缓存，将按线程优先在TLAB上分片，TLAB只是起缓存作用减少高并发下CAS带来的性能损失，跟GC的分代没有冲突。

2、当分配一个对象的时候会优先在Eden区域分配，如果Eden有足够的空间，那么内存分配很顺利的结束，不会触发任何GC操作；

3、当Eden区域空间不足的时候，会尝试着进行一次Minor GC，之所以说尝试是因为在进行Minor GC之前，虚拟机会检查老年代最大可用的连续空间是否大于新生代所有对象空间总和，如果是的那么可以保证这次Minor GC顺利进行；否则，虚拟机会检查HandlePromotionFailure这个参数是否设置为允许担保失败，如果允许那么虚拟机会根据经验值（这个经验值是历次晋升到老年代对象的平均大小）来决定是否尝试这次GC，如果小于或者JVM觉得不能冒险，那么会进行一次Full GC；

4、Minor GC时会采用复制算法将所有存活的对象复制到Survivor空间中（既包括Eden区域存活的对象，也包括另外一个Survivor存活下来的对象），如果这时发现Survivor空间不足，那么这些存活对象会直接进入老年代，这就是“空间分配”担保，前面说到冒险，是因为老年代的空间仍有可能不够，这时还是要进行一次Full GC，但是除了极端情况，大部分时候通过担保还是能有效避免频繁的Full GC的，如果Full GC后仍然没有足够空间，那只能抛出OutOfMemoryError；

5、对象在Eden空间出生，经过第一次Minor GC后能够顺利的被转移到Survivor的话，那么它的GC年龄就变成1，以后每在Survivor中熬过一次Minor GC，年龄就增加1，直到超过一定程度（-XX:MaxTenuringThreshold，默认15岁）则晋升到老年代；

6、规则是死的，人是活的，虚拟机开发人员还想到了一个“动态对象年龄判定”算法：如果Survivor区域中相同年龄所有对象大小总和超过Survivor空间的一半，年龄大于等于该年龄的对象就可以直接进去老年代；

7、对象也可以直接分配在老年代，这主要是针对那些大对象，因为大对象的内存分配代价比较大（需要连续的内存空间），所以JVM提供了-XX:PretenureSizeThreshold这个参数。

JVM采用分代垃圾回收。在JVM的内存空间中把堆空间分为年老代和年轻代。将大量（据说是90%以上）创建了没多久就会消亡的对象存储在年轻代，而年老代中存放生命周期长久的实例对象。年轻代中又被分为Eden区(圣经中的伊甸园)、和两个Survivor区。新的对象分配是首先放在Eden区，Survivor区作为Eden区和Old区的缓冲，在Survivor区的对象经历若干次收集仍然存活的，就会被转移到年老区

#### 11.(★★★)发生Full GC的情况下，如何dump出JVM内存，如何分析？

dump生成

dump可以是内存溢出时让其自动生成，或者手工直接导。配置jvm参数-XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath=/home/biapp/m.hprof

手工直接导，PID为进程号

jmap -dump:live,format=b,file=m.hprof PID

如何分析

eclipse转到memory analysis界面   
actons中Histogram列出每个类的实例数，Dominator Tree列出最大存活对象，Top Consumers打印出最高消耗对象，Leak Suspects包括泄露嫌疑人和系统概述   
Top Components热门组件：列出大于总堆的1％的组件的报告。

#### 12.(★★★)JVM内存区域由哪几部分组成

线程共享区域：

Java堆

方法区(包含常量池)

线程私有区域：

程序计数器

Java虚拟机栈

本地方法栈

具体解释 https://blog.csdn.net/x4609883/article/details/79759419

#### 13.(★★★)描述一下JVM加载class文件的原理机制？

Java中的所有类，都需要由类加载器装载到JVM中才能运行。类加载器本身也是一个类，而它的工作就是把class文件从硬盘读取到内存中。在写程序的时候，我们几乎不需要关心类的加载，因为这些都是隐式装载的，除非我们有特殊的用法，像是反射，就需要显式的加载所需要的类。

　　类装载方式，有两种

　　1.隐式装载， 程序在运行过程中当碰到通过new 等方式生成对象时，隐式调用类装载器加载对应的类到jvm中，

　　2.显式装载， 通过class.forname()等方法，显式加载需要的类

　　隐式加载与显式加载的区别：两者本质是一样?

Java类的加载是动态的，它并不会一次性将所有类全部加载后再运行，而是保证程序运行的基础类(像是基类)完全加载到jvm中，至于其他类，则在需要的时候才加载。这当然就是为了节省内存开销。

　Java的类加载器有三个，对应Java的三种类:（java中的类大致分为三种： 1.系统类 2.扩展类 3.由程序员自定义的类 ）

Bootstrap Loader // 负责加载系统类 (指的是内置类，像是String，对应于C#中的System类和C/C++标准库中的类)

|

- - ExtClassLoader // 负责加载扩展类(就是继承类和实现类)

|

- - AppClassLoader // 负责加载应用类(程序员自定义的类)

来源：https://www.cnblogs.com/Qian123/p/5707562.html

### 3.5 java代码优化

#### 1. （★★★）提高 Java 代码性能的各种技巧

Java 6,7,8 中的 String.intern – 字符串池

这篇文章将要讨论 Java 6 中是如何实现 String.intern 方法的，以及这个方法在 Java 7 以及 Java 8 中做了哪些调整。

字符串池

字符串池（有名字符串标准化）是通过使用唯一的共享 String 对象来使用相同的值不同的地址表示字符串的过程。你可以使用自己定义的 Map<String, String> （根据需要使用 weak 引用或者 soft 引用）并使用 map 中的值作为标准值来实现这个目标，或者你也可以使用 JDK 提供的 String.intern()。

很多标准禁止在 Java 6 中使用 String.intern() 因为如果频繁使用池会市区控制，有很大的几率触发 OutOfMemoryException。Oracle Java 7 对字符串池做了很多改进，你可以通过以下地址进行了解 <http://bugs.sun.com/view_bug.do?bug_id=6962931>以及 <http://bugs.sun.com/view_bug.do?bug_id=6962930>

Java 6 中的 String.intern()

在美好的过去所有共享的 String 对象都存储在 PermGen 中 — 堆中固定大小的部分主要用于存储加载的类对象和字符串池。除了明确的共享字符串，PermGen 字符串池还包含所有程序中使用过的字符串（这里要注意是使用过的字符串，如果类或者方法从未加载或者被条用，在其中定义的任何常量都不会被加载）

Java 6 中字符串池的最大问题是它的位置 — PermGen。PermGen 的大小是固定的并且在运行时是无法扩展的。你可以使用 -XX:MaxPermSize=N 配置来调整它的大小。据我了解，对于不同的平台默认的 PermGen 大小在 32M 到 96M 之间。你可以扩展它的大小，不过大小使用都是固定的。这个限制需要你在使用 String.intern 时需要非常小心 — 你最好不要使用这个方法 intern 任何无法控制的用户输入。这是为什么在 JAVA6 中大部分使用手动管理 Map 来实现字符串池

Java 7 中的 String.intern()

Java 7 中 Oracle 的工程师对字符串池的逻辑做了很大的改变 — 字符串池的位置被调整到 heap 中了。这意味着你再也不会被固定的内存空间限制了。所有的字符串都保存在堆（heap）中同其他普通对象一样，这使得你在调优应用时仅需要调整堆大小。这 个改动使得我们有足够的理由让我们重新考虑在 Java 7 中使用 String.intern()。

字符串池中的数据会被垃圾收集

没错，在 JVM 字符串池中的所有字符串会被垃圾收集，如果这些值在应用中没有任何引用。这是用于所有版本的 Java，这意味着如果 interned 的字符串在作用域外并且没有任何引用 — 它将会从 JVM 的字符串池中被垃圾收集掉。

因为被重新定位到堆中以及会被垃圾收集，JVM 的字符串池看上去是存放字符串的合适位置，是吗？理论上是 — 违背使用的字符串会从池中收集掉，当外部输入一个字符传且池中存在时可以节省内存。看起来是一个完美的节省内存的策略？在你回答这个之前，可以肯定的是你 需要知道字符串池是如何实现的。

在 Java 6，7，8 中 JVM 字符串池的实现

字符串池是使用一个拥有固定容量的 HashMap 每个元素包含具有相同 hash 值的字符串列表。一些实现的细节可以从 Java bug 报告中获得 <http://bugs.sun.com/view_bug.do?bug_id=6962930>

默认的池大小是 1009 (出现在上面提及的 bug 报告的源码中，在 Java7u40 中增加了)。在 JAVA 6 早期版本中是一个常量，在随后的 java6u30 至 java6u41 中调整为可配置的。而在java 7中一开始就是可以配置的（至少在java7u02中是可以配置的）。你需要指定参数 -XX:StringTableSize=N,  N 是字符串池 Map 的大小。确保它是为性能调优而预先准备的大小。

在 Java 6 中这个参数没有太多帮助，因为你仍任被限制在固定的 PermGen 内存大小中。后续的讨论将直接忽略 Java 6

Java 7 （直至 Java7u40）

在 Java7 中，换句话说，你被限制在一个更大的堆内存中。这意味着你可以预先设置好 String 池的大小（这个值取决于你的应用程序需求）。通常说来，一旦程序开始内存消耗，内存都是成百兆的增长，在这种情况下，给一个拥有 100 万字符串对象的字符串池分配 8-16M 的内存看起来是比较适合的（不要使用1,000,000 作为 -XX:StringTaleSize 的值 – 它不是质数；使用 1,000,003代替）

你可能期待关于 String 在 Map 中的分配 — 可以阅读我之前关于 HashCode 方法调优的经验。

你必须设置一个更大的 -XX:StringTalbeSize 值(相比较默认的 1009 ),如果你希望更多的使用 String.intern() — 否则这个方法将很快递减到 0 （池大小）。

我没有注意到在 intern 小于 100 字符的字符串时的依赖情况（我认为在一个包含 50 个重复字符的字符串与现实数据并不相似，因此 100 个字符看上去是一个很好的测试限制）

下面是默认池大小的应用程序日志：第一列是已经 intern 的字符串数量，第二列 intern 10,000 个字符串所有的时间（秒）

0; time = 0.0 sec

50000; time = 0.03 sec

100000; time = 0.073 sec

150000; time = 0.13 sec

200000; time = 0.196 sec

250000; time = 0.279 sec

300000; time = 0.376 sec

350000; time = 0.471 sec

400000; time = 0.574 sec

450000; time = 0.666 sec

500000; time = 0.755 sec

550000; time = 0.854 sec

600000; time = 0.916 sec

650000; time = 1.006 sec

700000; time = 1.095 sec

750000; time = 1.273 sec

800000; time = 1.248 sec

850000; time = 1.446 sec

900000; time = 1.585 sec

950000; time = 1.635 sec

1000000; time = 1.913 sec

测试是在 Core i5-3317U@1.7Ghz CPU 设备上进行的。你可以看到，它成线性增长，并且在 JVM 字符串池包含一百万个字符串时，我仍然可以近似每秒 intern 5000 个字符串，这对于在内存中处理大量数据的应用程序来说太慢了。

现在，调整 -XX:StringTableSize=100003 参数来重新运行测试：

50000; time = 0.017 sec

100000; time = 0.009 sec

150000; time = 0.01 sec

200000; time = 0.009 sec

250000; time = 0.007 sec

300000; time = 0.008 sec

350000; time = 0.009 sec

400000; time = 0.009 sec

450000; time = 0.01 sec

500000; time = 0.013 sec

550000; time = 0.011 sec

600000; time = 0.012 sec

650000; time = 0.015 sec

700000; time = 0.015 sec

750000; time = 0.01 sec

800000; time = 0.01 sec

850000; time = 0.011 sec

900000; time = 0.011 sec

950000; time = 0.012 sec

1000000; time = 0.012 sec

可以看到，这时插入字符串的时间近似于常量（在 Map 的字符串列表中平均字符串个数不超过 10 个），下面是相同设置的结果，不过这次我们将向池中插入 1000 万个字符串（这意味着 Map 中的字符串列表平均包含 100 个字符串）

2000000; time = 0.024 sec

3000000; time = 0.028 sec

4000000; time = 0.053 sec

5000000; time = 0.051 sec

6000000; time = 0.034 sec

7000000; time = 0.041 sec

8000000; time = 0.089 sec

9000000; time = 0.111 sec

10000000; time = 0.123 sec

现在让我们将吃的大小增加到 100 万（精确的说是 1,000,003）

1000000; time = 0.005 sec

2000000; time = 0.005 sec

3000000; time = 0.005 sec

4000000; time = 0.004 sec

5000000; time = 0.004 sec

6000000; time = 0.009 sec

7000000; time = 0.01 sec

8000000; time = 0.009 sec

9000000; time = 0.009 sec

10000000; time = 0.009 sec

如你所看到的,时间非常平均，并且与 “0 到 100万” 的表没有太大差别。甚至在池大小足够大的情况下，我的笔记本也能每秒添加1,000,000个字符对象。

我们还需要手工管理字符串池吗？

现在我们需要对比 JVM 字符串池和 WeakHashMap<String, WeakReference<String>> 它可以用来模拟 JVM 字符串池。下面的方法用来替换 String.intern：

private static final WeakHashMap<String, WeakReference<String>> s\_manualCache =

new WeakHashMap<String, WeakReference<String>>( 100000 );

private static String manualIntern( final String str )

{

final WeakReference<String> cached = s\_manualCache.get( str );

if ( cached != null )

{

final String value = cached.get();

if ( value != null )

return value;

}

s\_manualCache.put( str, new WeakReference<String>( str ) );

return str;

}

下面针对手工池的相同测试：

0; manual time = 0.001 sec

50000; manual time = 0.03 sec

100000; manual time = 0.034 sec

150000; manual time = 0.008 sec

200000; manual time = 0.019 sec

250000; manual time = 0.011 sec

300000; manual time = 0.011 sec

350000; manual time = 0.008 sec

400000; manual time = 0.027 sec

450000; manual time = 0.008 sec

500000; manual time = 0.009 sec

550000; manual time = 0.008 sec

600000; manual time = 0.008 sec

650000; manual time = 0.008 sec

700000; manual time = 0.008 sec

750000; manual time = 0.011 sec

800000; manual time = 0.007 sec

850000; manual time = 0.008 sec

900000; manual time = 0.008 sec

950000; manual time = 0.008 sec

1000000; manual time = 0.008 sec

当 JVM 有足够内存时，手工编写的池提供了良好的性能。不过不幸的是，我的测试（保留 String.valueOf(0 < N < 1,000,000,000)）保留非常短的字符串，在使用 -Xmx1280M 参数时它允许我保留月为 2.5M 的这类字符串。JVM 字符串池 (size=1,000,003）从另一方面讲在 JVM 内存足够时提供了相同的性能特性，知道 JVM 字符串池包含 12.72M 的字符串并消耗掉所有内存（5倍多）。我认为，这非常值得你在你的应用中去掉所有手工字符串池。

在 Java 7u40+ 以及 Java 8 中的 String.intern()

Java7u40 版本扩展了字符串池的大小（这是组要的性能更新）到 60013.这个值允许你在池中包含大约 30000 个独立的字符串。通常来说，这对于需要保存的数据来说已经足够了，你可以通过 -XX:+PrintFlagsFinal JVM 参数获得这个值。

我尝试在原始发布的 Java 8 中运行相同的测试，Java 8 仍然支持 -XX:StringTableSize 参数来兼容 Java 7 特性。主要的区别在于 Java 8 中默认的池大小增加到 60013：

50000; time = 0.019 sec

100000; time = 0.009 sec

150000; time = 0.009 sec

200000; time = 0.009 sec

250000; time = 0.009 sec

300000; time = 0.009 sec

350000; time = 0.011 sec

400000; time = 0.012 sec

450000; time = 0.01 sec

500000; time = 0.013 sec

550000; time = 0.013 sec

600000; time = 0.014 sec

650000; time = 0.018 sec

700000; time = 0.015 sec

750000; time = 0.029 sec

800000; time = 0.018 sec

850000; time = 0.02 sec

900000; time = 0.017 sec

950000; time = 0.018 sec

1000000; time = 0.021 sec

总结

由于 Java 6 中使用固定的内存大小（PermGen）因此不要使用 String.intern() 方法

Java7 和 8 在堆内存中实现字符串池。这以为这字符串池的内存限制等于应用程序的内存限制。

在 Java 7 和 8 中使用 -XX:StringTableSize 来设置字符串池 Map 的大小。它是固定的，因为它使用 HashMap 实现。近似于你应用单独的字符串个数（你希望保留的）并且设置池的大小为最接近的质数并乘以 2 （减少碰撞的可能性）。它是的 String.intern 可以使用相同（固定）的时间并且在每次插入时消耗更小的内存(同样的任务，使用java WeakHashMap将消耗4-5倍的内存)。

在 Java 6 和 7（Java7u40以前） 中 -XX:StringTableSize 参数的值是 1009。Java7u40 以后这个值调整为 60013 （Java 8 中使用相同的值）

如果你不确定字符串池的用量，参考：-XX:+PrintStringTableStatistics JVM 参数，当你的应用挂掉时它告诉你字符串池的使用量信息。

### 3.6其他

#### 1.(★)请写出常用的linux指令，不低于10个，请写出linux tomcat启动停止命令

答：

ls

cd

rm

mkdir

cat

vi

mv

cp

touch

pwd

find

Tomcat 启动：sh startup.sh

Tomcat 停止：sh stopup.sh

#### 2.(★★)Memcached服务的内存如何分配？集群如何管理？

Memcached的对内存的管理是以slab为单元进行管理的。它是一次申请内存的最小单位。每个slab class对应着一个或者多个chunk。chunk是存放元素的最小单元。

memcached会根据元素的大小，将其放到合适的slab class中，每个slab class 中chunk的大小是一样的。所以元素放到chunk内存空间后，可能会有部分剩余空间。它是根据最优选择的。

slab，是厚片的意思。而chunk是厚厚的一块的意思。

#### 3.(★★)除Java之外还涉猎哪些技术，这些技术有哪些优缺点。

Python C语言

Python的优点

1、python非常简单，非常适合人类阅读。阅读一个良好的Python程序就感觉像是在读英语一样，尽管这个英语的要求非常严格！

2、易学。python虽然是用c语言写的，但是它摈弃了c中非常复杂的指针，简化了python的语法

3、可移植性————由于它的开源本质，Python已经被移植在许多平台上（经过改动使它能够工作在不同平台上）。

4、可扩展性和可嵌入性。如果你需要你的一段关键代码运行得更快或者希望某些算法不公开，你可以把你的部分程序用C或C++编写，然后在你的Python程序中

5、丰富的库。

6、Python确实是一种十分精彩又强大的语言。它合理地结合了高性能与使得编写程序简单有趣的特色。

7、规范的代码。Python采用强制缩进的方式使得代码具有极佳的可读性。

缺点

1、运行速度，有速度要求的话，用C++改写关键部分吧。不过对于用户而言，机器上运行速度是可以忽略的。因为用户根本感觉不出来这种速度的差异。

2、既是优点也是缺点，python的开源性是的Python语言不能加密，但是目前国内市场纯粹靠编写软件卖给客户的越来越少，网站和移动应用不需要给客户源代码，所以这个问题就是问题了。国随着时间的推移，很多国内软件公司，尤其是游戏公司，也开始规模使用他。

3、构架选择太多（没有像C#这样的官方.net构架，也没有像ruby由于历史较短，构架开发的相对集中。Ruby on Rails 构架开发中小型web程序天下无敌）。不过这也从另一个侧面说明，python比较优秀，吸引的人才多，项目也多。

C语言

c语言的优点：

1、c语言具有很高的运行效率。

2、c语言具有简洁紧凑、使用灵活的语法机制，并能直接访问硬件。

3、c语言具有出色的可移植性，能在多种不同体系结构的软/硬件平台上运行。

c语言的缺点：

1、c语言代码的复用性差。

2、c语言代码的维护性差。

3、c语言代码的扩展性差。

#### 4.(★★★)谈谈自己的技术成长线路及未来规划。

#### 5.(★)Git里fetch和pull的区别是什么？

git pull会拉取服务器的代码并自动合并到本地

git fetch会拉取服务器的代码，但是不会进行合并

#### 6.(★)认证（authentication）和鉴权（authorization）是什么区别？

举例说明：你要登陆论坛，输入用户名张三，密码1234，密码正确，证明你张三确实是张三，这就是 authentication；再一check用户张三是个版主，所以有权限加精删别人帖，这就是 authorization

#### 7.(★★)请举例几个Linux常用命令，并说明他们的功能

ls -a 列出目录所有文件，包含以.开始的隐藏文件

ls -A 列出除.及..的其它文件

ls -r 反序排列

ls -t 以文件修改时间排序

ls -S 以文件大小排序

ls -h 以易读大小显示

ls -l 除了文件名之外，还将文件的权限、所有者、文件大小等信息详细列出来

#### 8.(★★)列出不少于5个查看Linux系统信息或运行状态的常用命令？

linux CPU大小

cat /proc/cpuinfo |grep "model name" && cat /proc/cpuinfo |grep "physical id"

内存大小

cat /proc/meminfo |grep MemTotal

硬盘大小

fdisk -l |grep Disk

查看内核/操作系统/CPU信息的linux系统信息命令

uname –a

查看CPU信息的linux系统信息命令

cat /proc/cpuinfo

#### 9.(★★★)如何让程序在后台运行，以及如何然程序开机启动？

程序在后台运行

Nohup

Setsid

程序开机启动

一、/etc/rc.local

这是一个最简单的方法，编辑“/etc/rc.local”，把启动程序的shell命令输入进去即可，类似于windows下的“启动”。

例如，每次开机时要执行一个haha.sh，这个脚本放在/opt下面，那就可以在“/etc/rc.local”中加一行“/opt/./haha.sh”，或者两行“cd /opt”和“./haha.sh”。

二、crontab

通过crontab可以设定程序的执行时间表，例如让程序在每天的8点，或者每个星期一的10点执行一次。

三、注册系统服务

操作系统自带的服务，如ssh，ftp等等，开机都是自动启动的，我们也可以通过这种方式让自己开发的程序提高“身价”

来源：https://blog.csdn.net/jiafu1115/article/details/54629749/

#### 10.(★)如何输出系统时间到test.log文件，使用>和>>有什么不同？

echo `date +"%Y-%m-%d %H:%M:%S"` begin >> ./a.log

>表示覆盖原文件内容，>>表示追加内容

#### 11.(★)改变文件所有者的命令为（C）

A chmod B touch C chown D cat

#### 12.(★)与命令ls-l -f wls等价的命令是（A）

A ls -F -l wsl B ls-LF wls C ls\_FL wls D ls wls -l

#### 13.(★)你使用mkdir命令创建一个临时文件夹/tmp/aaa，并将一些文件复制其中。使用完后要删除/mnt/tmp文件夹及其中的所有文件应该使用命令（B）

A rm /tmp/aaa B rm -r /tmp/aaa C rmdir -r /tmp/aaa Drmdir /tmp/aaa

#### 14.(★)linux文件系统的目录结构是一个倒挂的树，文件都按其作用分门别类地放在相关的目录中现有一个外部设备文件，我们应该将其放在（C）目录中

A /bin B /etc C /dev D /lib