******

一、 Core Java Interview Questions

1. ******What are the core OOP's concepts?

Abstraction, Encapsulation,Inheritance and Polymorphism are the core OOP's concepts.

2. ******What is an Object?

Object is an instance of a class. It has state, behaviour and identity. It is also called as an instance of a class.

3. ******What is meant by Encapsulation?

Encapsulation is the technique of making the fields in a class private and providing access to the fields via public methods.

- The fields of a class can be made read-only or write-only.
- Data hiding.

4. ******What is the difference between an object and object reference?

An **object** is an instance of a class. **Object reference** is a pointer to the object. There can be many refernces to the same object.

5. What will trim() method of String class do?

trim() eliminate spaces from both the ends of a string.

6. ******What is the difference between String, Stringbuffer and StringBuilder?

String is immutable whereas **StringBuffer** and **StringBuilder** can change their values.
The only difference between **StringBuffer** and **StringBuilder** is that **StringBuilder** is unsynchronized whereas **StringBuffer** is synchronized. So when the application needs to be run only in a single thread then it is better to use **StringBuilder**. **StringBuilder** is more efficient than **StringBuffer**.

7. What is the difference between Path and Classpath?

Path and Classpath are operating system level environment variales. **Path** is used define where the system can find the executables(.exe) files and **classpath** is used to specify the location .class files.

8. What are the default values of primitive data types?

For boolean data type, it is false. For byte,short,int and long, it is 0. For float and double, it is 0.0. For **char**, the default value is 'u000'.

9. Can char be manipulated like integers?

Yes possible. The below is an example.char ch = 'A';System.out.println(ch++); The above statement will print B. ++ is a numeral operand and since char is internally represented as integer, ++ operand can be applied on char value.

10. What are different ways to declare an array? 笔试题应该

Object obj[]; // most frequently used form of array declaration.Object []obj; // nothing wrong with this. Object[] obj; // nothing wrong with this. int i[][]; // most frequently form multi-dimensional array int[]

i[]; // nothing wrong with this. int[] i[],j; // nothing wrong with this. Important : i is double dimensional whereas j is single dimensional.

11. ******What is the difference between static and non-static variables?

A **static** variable is associated with the class as a whole rather than with specific instances of a class. **Non-static** variables take on unique values with each object instance

12. Why is the main method declared static?

main method is the entry point for a Java application and main method is called by the JVM even before the instantiation of the class hence it is declared as static. static methods can be called even before the creation of objects.

13. Can a main method be overloaded?

Yes. You can have any number of main methods with different method signature and implementation in the class.

14. Does the order of public and static declaration matter in main method?

No it doesn't matter but void should always come before main().

15. Can a source file contain more than one Class declaration?

Yes. A single source file can contain any number of Class declarations but only one of the class can be declared as public.

16. ******Explain Inheritance?

Inheritance is a concept where the properties and behavior of a class is **reused** in another class. The class which is being reused or inherited is called the **super class** or the **parent class**. The class which is inheriting is called the subclass or the child class of the inherited class.

17. Can a subclass be referenced for a super class object?

No. If Vehicle is super class and Car is the subclass then the following reference would be wrong – Car c = new Vehicle();（这个是错的）

18. Can a parent class be referenced for a subclass object?

Yes. The below is an example :Vehicle v = new Car();

19. Can an interface be referenced for any object?

Yes. We can create a object reference for an interface. The object should have provided the implementation for the object. **Runnable r = new Test();** Test class should have implemented the runnable interface and overridded the run method.

20. ******What are constructors?

Constructors are used to initialize an object. Constructors are invoked when the new operator on the class are called.

21. What is the sequence of constructor invocation?

When you instantiate a subclass, the object will begin with an invocation of the constructor in the base class and **initialize downwards** through constructors in each subclass till it reaches the final subclass constructor. Note: This top down imagery for class inheritance, rather than a upward tree-like approach, is standard in OOP but is sometimes confusing to newcomers.

22. ******Explain Polymorphism?

The dictionary definition of **polymorphism** refers to a principle in biology in which an organism or species can have many different forms or stages. This principle can also be applied to object-oriented programming and languages like the Java language. **Subclasses of a class can define their own unique behaviors and yet share some of the same functionality of the parent class. Polymorphism** is the capability of an action or method to do different things based on the object that it is acting upon. Overloading and overriding are two types of polymorphism.

23. ******What is Overloading?

In a class, if two methods have the same name and a different argument, it is known as overloading in Object oriented concepts. 可以在一个类中

24. ******Explain Overriding?

When a class defines a method using the **same name**, **return type**, and **arguments** as a method in its **superclass**, the method in the class **overrides** the method in the superclass. When the method is invoked for an object of the class, it is the new definition of the method that is called, and not the method definition from superclass.子类和父类间的

Rules to follow :

Methods may be overridden to be **more public**, **not more private.**

Methods may be overridden to be **throws** the **same exception** or **subclass** of the **exception** thrown by the method of superclass

25. Can a class declared as private be accessed outside it's package?

X A class can't be declared as private. It will result in compile time error?

Classes and interfaces **can** be **declared** private or **protected** within their enclosing **classes**.

A **class** which is local to a block is not a member, and so cannot be public, private , **protected** , or static.

reflection can be used to bypass the java access rules.

26. *******Can a class be declared as protected?

X A class can't be declared as protected. only **methods** can be **declared** as **protected**.

Classes and interfaces **can** be **declared** private or **protected** within their enclosing **classes**.

A **class** which is local to a block is not a member, and so cannot be public, private , **protected** , or static.

reflection can be used to bypass the java access rules.

27. ******What is the access scope of a protected method?

A **protected method** can be accessed by the classes within the **same package** or by the **subclasses** of the class in any package.

28. What is the impact of marking a constructor as private?

Nobody can instantiate the class from outside that class.

Traditional uses is used for singleton pattern.

29. What is meant by default access?

Default(Friendly) access means the class, method, construtor or the variable can be accessed only **within** the **package**.

30. Can a final variable be declared inside a method?

X **No**. **Local** variables **cannot** be declared as **final**.
Yes, final variable can be declared inside a method.

31. What is the impact of declaring a method as final?

A method declared as final **can't** be **overridden**. A sub-class doesn't have the independence to provide different implementation to the method. A method can be overloaded.

32. How to define a constant variable in Java?

The variable should be declared as static and final. So only one copy of the variable exists for all instances of the class and the value can't be changed also. static final int PI = 3.14; is an example for constant.

33. ******What is static in java?

Static means one per class, not one for each object no matter how many instance of a class might exist. This means that you can use them without creating an instance of a class.
Static methods are implicitly final, because overriding is done based on the type of the object, and **static methods** are attached to a class, not an object. **A static method** in a superclass can be shadowed(override) by another static method in a subclass, as long as the original method was not declared final. However, you **can't override** a static method with a **non-static** method. In other words, you can't change a static method into an instance method in a subclass.
Static varibales are not serialized.

34. Can a class be declared as static?

No a class cannot be defined as static in fact every class is "static". Only a method, a variable or a block of code can be declared as static. But inner class can be static!

35. When will you define a method as static?

When a method needs to be accessed even **before** the **creation** of the **object** of the class then we should declare the method as static.

36. The difference between static and final

Static variable is a variable shared by all the instances of objects and it has only single copy. Final variable is a constant variable and it can't be changed.

37. Can we declare a static variable inside a method?

Static varaibles are **class level** variables and they **can't** be declared inside a method. If declared, the class will not compile.

38. ******What is an Abstract Class and what is it's purpose?

A Class which **doesn't provide** complete **implementation** is defined as an **abstract class**. It serves **as a template**. A class that is abstract may **not be instantiated** (ie, you may not call its constructor), abstract class may **contain static data**. Any class with an **abstract method** is automatically **abstract itself**, and must be declared as abstract. A class may be declared abstract even if it has no abstract methods. This prevents it from being instantiated

39. ******What is an Interface?

Interfaces say what a class must do but does not say how a class must do it. Interfaces are 100% abstract. Declare method without the implementation.

40. ******What is the difference between interface and abstract class?

- **interface** contains methods that must be abstract; **abstract** class may contain concrete methods.
- **interface** contains **variables** that must be static and final; **abstract** class may contain non-final and final variables.
- members in an **interface** are public by default, **abstract** class may contain non-public members.
- **interface** is used to "implements"; whereas **abstract** class is used to "extends".
- **interface** can be used to achieve multiple inheritance; **abstract** class can be used as a single inheritance.
- **interface** can "extends" another interface, **abstract** class can "extends" another class and "implements" multiple interfaces.
- **interface** is absolutely abstract; **abstract** class can be invoked if a main() exists.
- **interface** is more flexible than **abstract** class because one class can only "extends" one super class, but "implements" multiple interfaces.
- If given a choice, use **interface** instead of **abstract** class.

41. Can a abstract class be defined without any abstract methods?

Yes it's possible. This is basically to avoid instance creation of the class.

42. What happens if a subclass has inherited a abstract class but has not provided implementation for all the abstract methods of the super class?

Then the subclass also becomes abstract. This will be enforced by the compiler.

43. What happens if a class has implemented an interface but has not provided implementation for a method in a interface?

Its the same as the earlier answer. The class has to be marked as abstract. This will be enforced by the compiler.

44. Can you create an object of an abstract class?

Not possible. Abstract classes are not concrete and hence can't be instantiated. If you try instantiating, you will get compilation error.

45. Can I create a reference for an abstract class?

Yes. You can create a reference for an abstract class only when the object being provided the implementation for the abstract class - it means that the object should be of a concrete subclass of the abstract class. This applies to interfaces also. Below is an example for interface referencing an **object:java.sql.Connection con = java.sql.DriverManager.getConnection("");**

46. Can a class be marked as native?

No. Only **methods can** be marked as **native**.
The native keyword is applied to a method to indicate that the method is implemented in native code using JNI(Java Native Interface).

47. ******What is the disadvantage of native methods?

Since Java runs faster, native method is less common.
By using **native methods**, the java program **loses platform independence** - the accessed platform might be tightly coupled with an operating system hence java program also loses OS independence.
Potential security risk， Loss of portability.

48. Can a method inside an Interface be declared as final?

Not possible. Doing so will result in compilation error. Final method can not be override. public and abstract are the only applicable modifiers for method declaration in an interface.

49. Why is an Interface able to extend more than one Interface but a Class can't extend more than one Class?

Basically **Java doesn't allow multiple inheritance,** so a Class is restricted to extend only one Class. But an Interface is a pure abstraction model and doesn't have inheritance hierarchy like classes (do remember that the base class of all classes is Object). So an Interface is allowed to extend more than one Interface.

50. Can an Interface be final?

Not possible. Doing so will result in compilation error. Final method can not be override.

51. Can a class be defined inside an Interface?

Yes it's possible.

52. Can an Interface be defined inside a class?

Yes it's possible.

53. ******What is a Marker Interface?

? An **Interface** which **doesn't** have **any declaration** inside but still enforces a mechanism.

In earlier versions of Java, Marker Interfaces were the only way to declare metadata about a class.
In modern Java, marker interfaces have no place. They can be completely replaced byAnnotations, which allow for a very flexible metadata capability.

54. ******What are the modifiers in Java ?

- **public** : Public class is **visible in other packages**, field is visible everywhere (class must be public too)
- **private** : Private variables or methods may be used **only by an instance of the same class** that declares the variable or method, A private feature may only be accessed by the class that owns the feature.
- **protected** : Is available to **all classes in the same package** and also available to all subclasses of the class that owns the protected feature. This access is provided even to subclasses that reside in a different package from the class that owns the protected feature.
- **default** :What you get by default without any access modifier (ie, public private or protected).It means that it is **visible to all within a particular package**

55. Can we define private and protected modifiers for variables in interfaces?

No. Always all variables declared inside an interface are of public access.

56. What modifiers are allowed for methods in an Interface?

Only public and abstract modifiers are allowed for methods in interfaces.
In this case, an interface is intended to serve as a public protocol for communicating with an object,

57. When can an object reference be cast(上溯) to an interface reference?

An object reference can be cast to an interface reference **when** the object **implements** the referenced interface.

58. ******What are Nested Classes?

？！ **Static Inner classes** are called sometimes referred as **nested classes** because these classes can exist without any relationship with the containing class.

Outer.Nested instance = new Outer.Nested(); In Java a **static** nested class is essentially a normal class that has just been nested inside another class. Being **static**, a **static** nested class can only access instance variables of the enclosing class via a reference to an instance of the enclosing class

59. ******What are the disadvantages of Inner classes?

- **Inner classes** are **not reusable** hence defeats one of the Encapsulation feature of Java.
- Highly **confusing** and **difficult** syntax which leads poor code maintainability.

60. ******What is difference between shallow copy and deep copy ?

Only instances of classes that implement the **Cloneable interface** can be cloned. Trying to clone an object that does not implement the Cloneable interface throws a CloneNotSupportedException. Both shallow copy and deep copy object need to implement Cloneable Interface.
Shallow copy：浅拷贝就是被复制的是对象的 reference
Deep Copy: 被复制的对象复制了原对象的值，并放入了一个新的地址中 ，相当于完全新建了一个跟之前一样的对象

61. ******Why to override equals() and hashCode()? and How can I implement both equals() and hashCode() for Set ?

- If you are implementing **HashSet** to store **unique** object then you need to implement equals() and hashcode() method.
- if two **objects** are **equal** according to the equals() method, they **must** have the **same hashCode()** value (although the reverse is not generally true).相同的两个对象有相同的 hashcode，可以用来比较两个对象是否相同
- 因为是用 hashcode 来比较两个是否相等的，所以要 override hashCode();

62. ******What is the difference between int and Interger? You can have question like why we need Wrapper Class(Integer)?

int is primitive type and Integer is **Wrapper Class**; if you are using Collection objects like Hashtable etc. you can't use int as a key. you have to use object so you can use Integer class

63. ******What is serialization? 这题很重要，要跟着下题一起回答

Serialization involves **saving** the **current state** of an object **to a stream**, and **restoring** an **equivalent object from** that **stream**. （For RMI use between two different end point, 只能对对象 serialize 不能对 base class）

(Better Answer) Serialization is the process of converting an object into a stream of bytes in order to store the object or transmit it to memory, a database, or a file. Its main purpose is to save the state of an object in order to be able to recreate it when needed. The reverse process is called deserialization.

For an object to be serialized, it **must** be an **instance of a class** that implements either the Serializable or Externalizable interface. Both interfaces only permit the saving of data associated with an object's variables.

They depend on the class definition being available to the Java Virtual Machine at reconstruction time in order to construct the object.

简单说就是为了保存在**内存中**的各种对象的状态（也就是实例变量，不是方法），并且可以把保存的对象状态再读出来。虽然你可以用你自己的各种各样的方法来保存 object states，但是 Java 给你提供一种应该比你自己好的保存对象状态的机制，那就是序列化。

- **什么情况下需要序列化**
  - a）当你想把的内存中的对象状态保存到一个文件中或者数据库中时候；
  - b）当你想用套接字在网络上传送对象的时候；
  - c）当你想通过 RMI 传输对象的时候；

64. When you serialize an object, what happens to the object references included in the object?

The serialization mechanism generates an object graph for serialization. Thus it **determines whether** the **included object references are serializable or not.** This is a **recursive process**. **Thus** when an object is serialized, all the included objects are also serialized along with the original obect.

65. ******What happens to the static fields of a class during serialization?

There are **three exceptions** in which serialization does **not** necessarily **read** and **write to the stream**. These are

这是三个不需要读写 stream 的

1. Serialization **ignores** static fields, because they are not part of any particular state.
2. **Base class** fields are **only handled** if the base class itself **is serializable**.
3. **Transient fields.**

66. What is the difference between declaring a variable and defining a variable?

In **declaration** we just mention the type of the variable and it's name. We do **not initialize it**. But **defining** means **declaration + initialization.**

二、 Exception Handling Interview Questions & Answers

1. ******What is the difference between Exception and Error in java?

Exception and **Error** are the subclasses of the **Throwable** class. **Exception** class is used for exceptional conditions that user program should catch. **Error** defines exceptions that are not excepted to be caught by the user program. Example is Stack Overflow.

2. ********Differentiate between Checked Exceptions and Unchecked Exceptions?**

- A **checked exception** is some subclass of Exception (or Exception itself), **excluding** class **RuntimeException** and its subclasses. Making an exception checked **forces client programmers to deal with** the possibility that the exception will be thrown. eg, IOException thrown by java.io.FileInputStream's read() method·一定要进行处理的
- **Unchecked exceptions** are **RuntimeException** and any of its subclasses. Class **Error** and its subclasses **also** are **unchecked**. With an unchecked exception, **however**, the compiler **doesn't force** client programmers either to catch the exception or declare it in a throws clause. In fact, client programmers may not even know that the exception could be thrown. eg, StringIndexOutOfBoundsException thrown by String's charAt() method· Checked exceptions must be caught at compile time. Runtime exceptions do not need to be. Errors often cannot be 不一定要进行处理
- **Checked & Unchecked Exception functions not so much difference. It is only programmers habit or community displine to decide wheter to use Unchecked exception or Checked exception.**

3. ********What is the difference between throw and throws?**

throw is used to explicitly raise a exception within the program, the statement would be throw new Exception(); **throws** clause is used to indicate the exceptions that are not handled by the method. It must specify this behavior so the callers of the method can guard against the exceptions. **throws** is specified in the method signature. If multiple exceptions are not handled, then they are separated by a comma. the statement would be as follows: public void doSomething() throws IOException,MyException{} 就是 throw 抛一个，throws 抛一堆

4. ********What is the importance of finally block in exception handling?**

Finally block will be executed whether or not an exception is thrown. If an exception is thrown, the finally block will execute even if no catch statement match the exception. Any time a method is about to return to the caller from inside try/catch block, via an uncaught exception or an explicit return statement, the finally block will be executed. Finally is used to free up resources like database connections, IO handles, etc. （**System.exit()** or **JVM crashes “**finally” will not execute.）

5. **Can a finally block exist with a try block but without a catch?**

Yes. The following are the combinations **try/catch** or **try/catch/finally** or **try/finally.**

6. What will happen to the Exception object after exception handling?

Exception object will be **garbage collected**.

7. Exceptions can be caught or rethrown to a calling method. True/False?

True.

8. ******What is the purpose of finalization?

Finalization invokes finalized() method.
The purpose of **finalization** is to give an **unreachable object** the opportunity to perform some actions **before** the object is garbage collected.

9. ******How does finally block differ from finalize() method?

Finally block will be executed whether or not an exception is thrown. So it is used to free resources.
finalize() is a protected method in the Object class which is called by the JVM just **before** an object is garbage collected.

What are the different ways to handle exceptions?
There are two ways to handle exceptions,
By **wrapping** the desired code **in** a **try block** followed by a catch block to catch the exceptions.
List the desired exceptions in the **throws clause** of the method and let the caller of the method handle those exceptions.

Although "throws" is convient, but use try catch block we can locate the position where exception happens.

10 Name "**Throwable**" ?

By catching Throwable it **includes things with subclass Error**. You should generally not do that, except perhaps at the very **highest "catch all"** level of a thread where you want to log or otherwise handle absolutely everything that can go wrong. It would be more typical in a framework type application (for example an application server or a testing framework) where it can be running unknown code and should not be affected by *anything* that goes wrong with that code, as much as possible.

三、 Java Threads Interview Questions & Answers

1. ******What are the two types of multitasking?

a. **Process**-based. b. **Thread**-based.

2. ******What is a Thread?

A **thread** is a single sequential flow of control **within** a **program**.

3. ******The difference between thread and process

Both processes and threads are independent sequences of execution

Typical difference is that threads (of the same process) run in a shared memory space, while processes run in separate memory spaces.

- A **process** can contain more than one thread.
- A **process** is considered as "heavyweight" while a **thread** is deemed as "lightweight".
- **Processes** are heavily dependent on system resources available while **threads** require minimal amounts of resource.
- Modifying a main **thread** may affect subsequent **threads** while changes on a parent **process** will not necessarily affect child **processes**.
- **Threads** within a process **communicate** directly while **processes** do not communicate so easily.
- **Threads** are easy to create while **processes** are not that straightforward.

4. ******What are the two ways to create a new thread?

a) **Extend** the **Thread** class and override the run() method.
b) **Implement** the **Runnable** interface and implement the **run**() method.

5. ******Which methods calls the run() method?

start() method.

6. ******What are the states of a Thread ?

Four state: **Ready**, **Running**, **Waiting** and **Dead**.

7. ******What will notify() method do?

notify() method **moves randomlly** a **thread** out of the **waiting pool** to **ready state**, but there is no guaranty which thread will be moved out of the pool.

8. ******What is the difference between sleep() and yield()?

When a Thread calls the **sleep**() method, it will return to its **waiting** state. When a Thread calls the **yield**() method, it returns to the **ready** state.

9. The difference between sleep() and wait()

sleep():

a) It is a **static** method on Thread class. It makes the current thread into the "Not Runnable" state for specified amount of time. B) During this time, the thread keeps the lock (monitors) it has acquired.

wait():

a)It is a method on Object class. It makes the current thread into the "Not Runnable" state. B) **wait** method must be called from synchronized context. C)**wait**() method releases the lock of object on which it has called

10. ******What is a Daemon Thread?

Daemon thread is a **low priority thread** which runs intermittently in the **back ground** doing the **garbage collection** operation for the java runtime system. **setDaemon** method is used to create a daemon thread We should call setDaemon(true) method on the thread object to make a thread as daemon thread.

11. ******What is the difference between normal thread and daemon thread?

Normal threads do mainstream activity, whereas **daemon threads** are used low priority work. Hence daemon threads are also stopped when there are no normal threads.

12. ******What does the start() method of Thread do?

The thread's **start**() method puts the thread **in ready** state and makes the thread **eligible to run**. start() method automatically calls the **run** () method.

13. ******What is synchronization and why is it important?

With respect to **multithreading**, **synchronization** is the capability to **control** the access of **multiple threads** to **shared resources**. Without synchronization, it is possible for one thread to modify a shared object while another thread is in the process of using or updating that object's value. This often leads to significant errors.

14. ******同步(synchronization)和异步(asynchronous)有何异同，在什么情况下分别使用他们？举例说明。

如果数据将在线程间共享。例如正在写的数据以后可能被另一个线程读到，或者正在读的数据可能已经被另一个线程写过了，那么这些数据就是共享数据，必须进行 ***synchronization*** 存取。当应用程序在对象上调用了一个需要花费很长时间来执行的方法，并且不希望让程序等待方法的返回时，就应该使用 ***asynchronous*** 编程，在很多情况下采用 **asynchronous** 途径往往更有效率。

15. ********What are synchronized methods and synchronized statements?**

Synchronized methods are methods that are used to control access to an object. A thread only executes a synchronized method after it has acquired the lock for the method's object or class. **Synchronized statements** （**Synchronized block?!**）are similar to synchronized methods. A synchronized statement can only **be executed after** a thread has **acquired** the **lock** for the object or class referenced in the synchronized statement

16. ********What are the two ways that a code can be synchronised? Can you declare a static method as synchronized?**

- **Method** can be declared as **synchronised**.
- A **block** of code be **sychronised**.

Yes, we can declare static method as synchronized. But the calling thread should acquire lock on the class that owns the method.

17. **Can a thread execute another objects run() method?**

Yes, A thread can execute it's own run() method or another objects run() method.

18. ********What is a deadlock?**

A condition that occurs when two processes are **waiting for each other** to complete **before proceeding**. The result is that both processes wait endlessly.

19. **What are all the methods used for Inter Thread communication and what is the class in which these methods are defined?**

a. wait(),notify() & notifyall() b. Object class

20. **The word synchronized can be used with only a method. True/ False?**

False. A block of code can also be synchronised.

21. **What is a Monitor?**

A **monitor** is an **object** which contains some synchronized code in it.

22. **What exception does the wait() method throw?**

The java.lang.Object class wait() method throws "**InterruptedException**".

23. ********What does notifyAll() method do?**

notifyAll() method moves **all waiting** threads from the **waiting** pool to **ready** state.

四、 Java Collections Interview Questions & Answers

1. ********什么是 Generic，或者说是 Generic 的优点**

Generic allow "**a type** or **method** to operate on objects of **various types** while providing compile-time type safety.

2. **<u>******Explain Iterator Interface.</u>**

An **Iterator** is similar to the **Enumeration** interface. With the **Iterator** interface methods, you can **traverse** a collection from start to end and **safely remove** elements from the underlying Collection. The **iterator**() method generally used in query operations. Basic methods: iterator.remove();iterator.hasNext();iterator.next();

3. **<u>******Explain Enumeration Interface.</u>**

The **Enumeration** interface allows you to iterate through all the elements of a collection. Iterating through an Enumeration is similar to iterating through an Iterator. **However**, there is **no removal support** with Enumeration.Basic methods: boolean hasMoreElements(); Object nextElement();

4. **<u>******What is the difference between Enumeration and Iterator interface?</u>**

The **Enumeration** interface allows you to iterate through all the elements of a collection. **Iterating** through an Enumeration is similar to iterating through an Iterator. **However**, there is **no removal support** (这个是唯一的区别)with **Enumeration**.

5. **<u>******Explain Set Interface.</u>**

In mathematical concept, a **set** is just a **group** of **unique items**, in the sense that the group contains **no duplicates**. The Set interface extends the Collection interface. Set does not allow duplicates in the collection. In Set implementations **null** is **valid** entry, but **allowed only once**.

6. **<u>******What are the two types of Set implementations available in the Collections Framework?</u>**

HashSet and **TreeSet** are the two Set implementations available in the Collections Framework.

7. **<u>******What is the difference between HashSet and TreeSet?</u>**

- **HashSet** Class implements java.util.Set interface to eliminate the duplicate entries and uses hashing for storage. Hashing is nothing but **mapping between** a key value and a data item, this provides **efficient searching**.
- The **TreeSet** Class implements java.util.Set interface provides an **ordered set**, eliminates duplicate entries and uses tree for storage.

8. **<u>******What is a List?</u>**

List is a **ordered** and **allow duplicated** collection of objects. The List interface extends the Collection interface. 可以重复的带排序的(可以用 Collections.sort()方法来进行排序)

9. **<u>What are the two types of List implementations available in the Collections Framework?</u>**

ArrayList and **LinkedList** are the two List implementations available in the Collections Framework.

10. **<u>******What is the difference between ArrayList, LinkedList and Vector?</u>**

- The **ArrayList Class** uses **array** for storage. An array storage are generally **faster** but we **cannot insert** and **delete** entries in middle of the list. To achieve this kind of addition and deletion requires a new array constructed. You can access any element at randomly.
- The **LinkedList Class** uses **linked** list for storage. A linked list **allows** elements to be **added**, **removed** from the collection at any location in the container by ordering the elements. With this implementation you can **only access** the elements in **sequentially**.
- **Vector** and **ArrayList** are very similar. **Both** of them represent a **growable array**. The **main difference** is that **Vector** is

synchronized(thread safe) while **ArrayList** is not. A **Vector** defaults to **doubling** the **size** of its array, while the **ArrayList** increases its array size by **50 percent**

11. Explain the functionality of Vector Class?

Once array size is set you cannot change size of the array. To deal with this kind of situations in Java uses **Vector**, it **grows** and **shrink** it's size automatically. Vector allows to store only **objects** not **primitives**. To store primitives, convert primitives in to objects using **wrapper classes** before adding them into Vector. The Vector reallocates and resizes itself automatically. (记住,**vector 要用 wrapper class** 的)

12. How do you store a primitive data type within a Vector or other collections class?

You need to wrap the primitive data type into one of the wrapper classes found in the java.lang package, like Integer, Float, or Double, as in:Integer in = new Integer(5);

13. ******Explain Map Interface.

A **map** is a special kind of set with **no duplicates.** The key values are used to lookup, or index the stored data. The Map interface is not an extension of Collection interface, it has its own hierarchy. Map does not allow duplicates in the collection. In Map implementations **null** is **valid entry**, but allowed **only once**.

14. What are the two types of Map implementations available in the Collections Framework?

HashMap and **TreeMap** are two types of Map implementations available in the Collections Framework.

15. ******What is the difference between HashMap and TreeMap?

- The **HashMap** Class uses hashing for storage. Indirectly Map uses Set functionality so, it does **not permit duplicates**.
- The **TreeMap** Class uses tree for storage. It provides the **ordered map**.

16. ******What is the difference between Hashtable and HashMap?

Both provide key-value access to data. The **key differences** are :

Hashtable is older: dictionary(inside null key exception)   Hashmap: improved hashtable

- **Hashtable** is synchronised but **HashMap** is not synchronised.
- **HashMap** permits once key multi-null values but **Hashtable** doesn't allow null values.
- iterator in the **HashMap** is fail-safe(it won't fail) while the iterator for the **Hashtable** is not fail safe.

17. How do I make an array larger?

You cannot directly make an array larger. You must **make a new (larger) array** and **copy** the original elements into it, usually with **System.arraycopy().** If you find yourself frequently doing this, the Vector class does this automatically for you, as long as your arrays are not of primitive data types.

18. Which is faster, synchronizing a HashMap or using a Hashtable for thread-safe access?

Because a synchronized HashMap requires an extra method call, a **Hashtable** is **faster** for synchronized access.

19. What is the use of Locale class?

The Locale class is used to **tailor program output** to the conventions of a particular geographic, political, or cultural region.

Locale 对象表示了特定的地理、政治和文化地区。需要 Locale
来执行其任务的操作称为语言环境敏感的 操作，它使用 Locale

为用户量身定制信息。例如，显示一个数值就是语言环境敏感的操作，应该根据用户的国家、地区或文化的风俗/传统来格式化该数值。

20. ******heap 和 stack 有什么区别。

栈是一种线形集合，其添加和删除元素的操作应在同一段完成。栈按照后进先出的方式进行处理。Later in earlier out。

堆是栈的一个组成元素

相同点：都是 RAM 中存放数据的地方

不同点：

- 栈 **stack**：存取速度快，但大小生命周期固定，主要应用于基本数据类型(primitives data type)（byte,int,long,float,double,char,Boolean）就是所有变量都存在 stack 里面
- 堆 **heap**：存取速度慢，但能动态分配内存，主要应用于对象（new 方式建立） 就是所有的对象都存在 heap 中

五、 Garbage Collection Interview Questions & Answers

1. ******Explain Garbage collection in Java?

In Java, **Garbage Collection(GC)** is **automatic**. Garbage Collector Thread runs as a low priority **daemon thread** freeing memory.

2. ******When does the Garbage Collection happen?

When there is **not enough memory**. Or when the **daemon GC thread**(**low priority)** gets a chance to **run**.

3. ******When is an Object eligible for Garbage collection?

An Object is eligible for GC, when there are **no references** to the **object**.

4. What are two steps in Garbage Collection?

- **Detection** of garbage collectible objects and **marking** them for garbage collection.
- **Freeing** the **memory** of objects marked for GC.

5. When is the finalize() called?

finalize() method is called by the GC **just before releasing** the object's memory. It is normally advised to release resources held by the object in finalize() method.

六、 Servlets Interview Questions & Answers（跟网络有关的也在其中）

1. ******What is a Servlet?

A **Servlet** is a **server side java program** which processes **client requests** and generates dynamic web content.

2. Explain the architechture of a Servlet?

javax.servlet.**Servlet** interface is the core abstraction which has to be implemented by all **servlets** either directly or indirectly. **Servlet** run **on** a **server side** JVM ie the servlet container. Most servlets implement the interface by extending either javax.servlet.**GenericServlet** or javax.servlet.http.**HTTPServlet**. A single servlet object serves **multiple requests** using **multithreading**.

3. ******Servlet Life Cycle ? servlet 的生命周期

The **life cycle** of a servlet is **controlled** by the **container** in which the servlet has been deployed. When a **request** is mapped to a servlet, the **container** performs the following steps. If an instance of the servlet does not exist, the Web container

1) **Loads** the servlet class. (装载)

2) **Creates** an **instance** of the servlet class. **Initializes** the servlet instance by calling the **init** method. **Initialization** is covered in Initializing a Servlet. （实例化）

3) **Invokes** the **service method**（**dopost(), doget()**）, passing a request and response object. Service methods are discussed in the section Writing Service Methods.（就绪， 调用）

4)If the container needs to **remove** the servlet, it finalizes the servlet by calling the servlet's **destroy method**.（销毁）

4. What is the difference between init() and init(ServletConfig) ?

- It is **BETTER** to use init(). If you use init() you have **no** such worries as calling **super**.**init**().
- If you use **init**(**servletconfig**) and **forgot** to **call super**.**init**(**config**) then servletconfig object will **not set** and you will not get the servletconfig object

5. ******What is the difference between an Applet and a Servlet?

An **Applet** is a **client side** java program that runs within a **Web browser** on the client machine whereas a **servlet** is a **server side** component which runs on the web server. An **applet** can use the user interface classes like AWT or Swing while the **servlet** does **not** have a user interface. **Servlet** waits for client's HTTP requests from a browser and **generates** a response that is displayed in the browser.

6. ******What is the difference between GenericServlet and HttpServlet?

- **GenericServlet** is a generalised and **protocol independent servlet** which defined in javax.servlet package. Servlets extending GenericServlet should **override service**() method.
- javax.servlet.http.HTTPServlet extends **GenericServlet**. **HTTPServlet** is **http protocol specific**. A subclass of HttpServlet must **override** at least one method of **doGet**(), doPost(),doPut(), doDelete(), init(), destroy() or getServletInfo().

7. ******What is the difference between doGet() and doPost()?

- **doGET Method** : Using get method we can able to pass 2K data（limited size） from HTML. All data we are passing to Server will be displayed in URL (request string).
- **doPOST Method** : In this method we does not have any size limitation. All data passed to server will be hidden, User cannot able to see this info on the browser.

8. ******What is the difference between ServletConfig and ServletContext?

- **ServletConfig** is a servlet configuration object used by a servlet container to pass information to a servlet during initialization. All of its initialization parameters can only be set in deployment descriptor. The **ServletConfig** parameters are specified for a particular servlet.

- **ServletContext** is an interface which defines a set of methods which the servlet uses to interact with its servlet container. **ServletContext** is common to all servlets within the same web application. Hence, servlets use **ServletContext** to share context information.

9. What is the difference between ServletContext and PageContext?

ServletContext: Gives the information about the **container**
PageContext: Gives the information about the **Request**

10. What is meant by a Web Application?

A Web Application is a **collection of servlets** and **content** installed under a specific subset of the server's URL namespace such as /catalog and possibly installed via a .war file.

11. What is the structure of the HTTP response?

The response can have **3 parts**:
1) Status Code - describes the status of the response. For example, it could indicate that the request was successful, or that the request failed because the resource was not available. If your servlet does not return a status code, the success status code, HttpServletResponse.SC_OK, is returned by default.
2) HTTP Headers - contains more information about the response. For example, the header could specify the method used to compress the response body.
3) Body - contents of the response. The body could contain HTML code, an image, etc.

12. ******What is a cookie?

A **cookie** is a bit of information that the **Web server sends to** the **browser** which then **saves** the **cookie** to a **file**. The browser sends the cookie back to the same server in every request that it makes. **Cookies are often used to keep track of sessions. Cookie** can be disable and not safe

13. ******What is the difference between session and cookie?

The difference between session and a cookie is two-fold.

- **session** should work regardless of the settings on the client browser. even if users decide to forbid the cookie (through browser settings) session still works. There is **no way to disable sessions from the client browser.**
- Javax.servlet.http.**Cookie** class has a setValue() method that accepts Strings.
- Javax.servlet.http.**HttpSession** has a setAttribute() method which takes a String to denote the name and java.lang.Object which means that **HttpSession** is capable of storing any java object. **Cookie** can only store String objects.

14. Which protocol will be used by browser and servlet to communicate?

HTTP

15. ******What is HTTP Tunneling（隧道）？

HTTP tunneling is used to **encapsulate** other **protocols** within the HTTP or HTTPS protocols. Normally the intra-network of an organization is blocked by a firewall and the network is exposed to the outer world only through a specific web server port, that listens for only HTTP requests. To use any other protocol, that bypasses the firewall, the protocol is embedded in HTTP and sent as HttpRequest. The masking of other protocol requests as http requests is **HTTP Tunneling.**

16. ********What's the difference between sendRedirect( ) and forward( ) methods?**

?! A **sendRedirect** method creates a **new** request (it's also reflected in browser's URL) where as **forward** method forwards the **same** request to the new target(hence the change is NOT reflected in browser's URL). The **previous** request scope objects are **no longer available** after a redirect because it results in a new request, but it's **available** in forward. sendRedirect is **slower** compared to forward. 就是说 sendredirect 用新建 request 而 forward 用老的

Control can be forward to resources available **within the server** from where the call is made. This transfer of control is done by the **container internally** and browser / client is not involved. This is the major difference between forward and sendRedirect. When the forward is done, the original request and response objects are transfered along with additional parameters if needed

Control can be redirect to resources to **different servers** or **domains**. This transfer of control task is **delegated** to the browser by the container. That is, the redirect sends a header back to the browser / client. This header contains the resource url to be redirected by the browser. Then the browser initiates a new request to the given url. Since it is a new request, the old request and response object is lost.

17. ********Is it true that servlet containers service each request by creating a new thread? If that is true, how does a container handle a sudden dramatic surge in incoming requests without significant performance degradation?**

The implementation depends on the Servlet engine. For **each request** generally, a **new Thread** is **created**. But to give performance boost, most **containers**, **create** and maintain a **thread pool** at the server startup time. To service a request, they simply borrow a thread from the pool and when they are done, return it to the pool. For this thread pool, upper bound and lower bound is maintained. **Upper bound** prevents the resource exhaustion problem associated with unlimited thread allocation. The lower bound can instruct the pool **not** to keep **too many idle threads**, freeing them if needed.

18. ******What is URL Encoding and URL Decoding?

URL encoding is the method of **replacing** all the spaces and other extra characters into their corresponding Hex Characters and **Decoding** is the reverse process converting all Hex Characters back their normal form.

19. ******Do objects stored in a HTTP Session need to be serializable? Or can it store any object?（serializable 可以在 rmi 里面找到，看看到时候是不是要移到这里来）之前也有 serialization 的问题

Yes, the objects need to be serializable, but **only if** your servlet container supports **persistent sessions**. Most lightweight servlet engines (like **Tomcat**) do not support this. However, many **EJB**-enabled servlet engines **do**. Even if your engine does support persistent sessions, it is usually possible to disable this feature.

20. ******Details about load on startup (load-on-startup tag) in Servlet ?

Used to **decide** whether servlet will be " **lazily** " or " **eagerly** " loaded Specified in web.xml file. If the value is not specified or is a Number < 0 then it means " **lazy loading** "

What " **lazy loading** " means is that servlet is **NOT** loaded by container on **startup**. **Servlet** in this case is **loaded** on **the first** client request - so the first client can experience **poor performance**
" **Eager** " **loading** means that the servlet is **initialised** on container startup. If there are two servelts A & B with values 0 & 1 than it means that Servlet A (having value = 0) will be loaded first
So if there are **more than one servlet** this element specifies the **order** of **loading** - **lower** integer values (including zero) are loaded **first.** If you specify this element but do not provide the value - even then the servlet will be the first servlet that gets loaded. To ensure that servlet follows " **lazy loading** " - do not provide this entry at all in web.xml file OR provide a negative number

七、 Struts Interview Questions

1. ******What is MVC?

Model-View-Controller (MVC) is a design pattern put together to help control change. MVC decouples interface from business logic and data.
Model : The model contains the core of the application's functionality. The model encapsulates the state of the application. Sometimes the only functionality it contains is state. It knows nothing about the view or controller.
View: The view provides the presentation of the model. It is the look of the application. The view can access the model getters, but it has no knowledge of the setters. In addition, it knows nothing about the controller. The view should be notified when changes to the model occur.
Controller: The controller reacts to the user input. It creates and sets the model.

2. Which design pattern the Interceptors in Struts2 is based on ?

Interceptors in Struts2 are based on Intercepting Filters.

3. ******What are the Advantages of Struts ? struts 的优点

Advantages of Struts :

Centralized Configuration : Rather than hard coding information into java programs, many Struts values are represented in XML or property files. This is organized. Your Action class , Form bean and JSP page information is in Struts_config.xml so don't need to search . All info in one place.
Form Beans : Don't need to set the form your value object . In the struts you don't need to do explicitly **request.getParameter().** Struts request processor will do for you. All the input data will be set to form bean.
Bean Tags : Struts provides a set of custom JSP tags (bean:write,in particular) that let you easily output the properties of JavaBeans components.
HTML tags : Struts provides a set of custom **JSP tags** to create **HTML** forms that are associated with JavaBeans components.
Form Field Validation(这点很关键) : Apache Struts has built-in capabilities for checking that form values are in the required format. If values are missing or in an improper format, the form can be automatically redisplayed with error messages and with the previously entered values maintained. This **validation** can be performed on the server (in Java),or both on the server and on the client (in JavaScript).

4. ******Life Cycle of Struts 2 就是 Struts2 的生命周期

1). Client Machine/Web Browser request for resource to server
2). Request reach at web.xml file, where we have entry of "Filter Dispatcher"
3). "Filter Dispatcher" calls to struts configuration file named "Struts.xml" and decides the suitable action.
4). Then the Interceptors use the required functions
5). After that the Action method executes all the functions like storing and retrieving data from a database.
6). Then Result will be sent according to the configuration file(Struts.xml)
7). it can be seen on the output of the browser in HTML, PDF, images or any other.
1、客户端浏览器发出 HTTP 请求。2、根据 web.xml 配置，该请求被 FilterDispatcher 接收。3、根据 struts.xml 配置，找到需要调用的 Action 类和方法， 并通过 IoC 方式，将值注入给 Aciton。4、Action 调用业务逻辑组件处理业务逻辑，这一步包含表单验证。5、Action 执行完毕，根据 struts.xml 中的配置找到对应的返回结果 result，并跳转到相应页面。6、返回 HTTP 响应到客户端浏览器。

5. What are the important sections in Struts Configuration File ? struts-config.xml?

The **five** important sections are:
1) **Form bean** definition section
2) **Global forward** definition section
3) **Action mapping** definition section
4) **Controller configuration** section
5) **Application Resources** definition section
******Are Interceptors and Filters different ? , If yes then how ?
Apart from the fact that both **Interceptors** and **filters** are based on intercepting filter, there are few differences when it comes to Struts2.
Filters: (1)Based on Servlet Specification (2)Executes on the pattern matches on the request.(3) Not configurable method calls
Interceptors: (1)Based on Struts2. (2)Executes for all the request qualifies for a front controller(A Servlet filter).And can be configured to execute additional interceptor for a particular action execution.(3)Methods in the Interceptors can be configured whether to execute or not by means of excludemethods or includeMethods. (see tutorial on this **Controlling Interceptor Behavior**)

6. In Struts1, the front-controller was a Servlet but in Struts2, it is a filter. What is the possible reason to change it to a filter?

There are 2 possibilities why filter is designated as front controller in Strtus2
(1) Servlet made as front controller needs developer to provide a right value in which lets the framework to initialize many important aspects(viz. struts configuration file)as the container starts. In absence of which the framework gets initialized only as the first request hits.Struts2 makes our life easy by providing front-controller as a filter and by nature the filters in web.xml gets initialized automatically as the container starts. There is no need of such load-on-startup tag.
(2). The second but important one is, the introduction of Interceptors in Struts2 framework. It not just reduce our coding effort, but helps us write any code which we would have used filters for coding and necessary change in the web.xml as opposed to Struts1.So now any code that fits better in Filter can now

moved to interceptors (which is more controllable than filters), all configuration can be controlled in struts.xml file, no need to touch the web.xml file.

7. What is the role of Action/ Model ?

Actions in Struts are **POJO** , is also considered as a **Model**. The role of **Action** is to execute business logic or delegate call to business logic by the means of **action** methods which is mapped to request and contains business data to be used by the view layer by means of setters and getters inside the **Action** class and finally helps the framework decide which result to render
An **Action** Class performs a role of an **adapter** between the contents of an **incoming HTTP request** and the corresponding **business logic** that should be executed to process this request.

8. What is Action Class?

An **Action class** in the struts application **extends** Struts 'org.apache.struts.action.Action" Class. **Action class** acts **as** wrapper around the business logic and **provides an inteface** to the application's Model layer. An **Action** works as an adapter between the contents of an incoming HTTP request and the business logic that corresponds to it. Then the struts **controller** (ActionServlet) selects an appropriate **Action** and **Request** Processor creates an instance if necessary, and finally calls **execute method** of **Action class**. To use the **Action**, we need to **Subclass** and **overwrite** the **execute**() **method**. and your business login in execute() method.
The **return type** of the execute method is **ActionForward** which is used by the Struts Framework to **forward** the request to the JSP as per the value of the returned ActionForward object. **ActionForward** JSP from **struts_config.xml** file.

9. ******What are Validators? and What are Basic Validators provided by the framework ?

Validator is a Java **class** that, when called by the Validator framework, executes a validation rule. The framework knows how to invoke a Validator class based on its method signature, as defined in a configuration file.
1) byte,short,integer,long,float,double
2) **creditCard** - Checks if the field is a valid credit card number.
3) **date** - Checks if the field is a valid date.
4) **email** - Checks if the field is a valid email address.
4) **mask** - Succeeds if the field matches the corresponding regular expression mask.
5) **maxLength** - Checks if the value's length is less than or equal to the given maximum length.
6) **minLength** - Checks if the value's length is greater than or equal to the given minimum length.
7) **range** - Checks if the value is within a minimum and maximum range.
8) **required** - Checks if the field isn't null and length of the field is greater than zero, not including whitespace.

10. ******What is the Benefits of Using the Validator framework in struts ?

A few of the **benefits** include:
1) A single **definition** of validation rules for an application.
2) Validation rules are **loosely coupled** to the application.
3) Server-side and client-side validation rules can be **defined** in **one location**.
4) **Configuration** of **new** rules and/or changes to existing rules are **made simpler**.

5) Supports **Internationalization**.
6) Supports **regular expressions**.
7) Can be used for both **Web-based** and **standard Java** applications.
8) Promotes a **declarative approach** rather than a programmatic one.

11. What is the relation between ValueStack and OGNL ?

A **ValueStack** is a place where all the data related to action and the action itself is stored. OGNL is a mean through which the data in the ValueStack is manipulated.

What is the difference between empty default namespace and root name space ?

If the namespace attribute is not defined in the package tag or assigned "" value then it is called empty default namespace. While if "/" is assigned as value to the namespace attribute then it is called as root namespace.

The root namespace is treated as all other explicit namespaces and must be matched. It’s important to distinguish between the empty default namespace, which can catch all request patterns as long as the action name matches, and the root namespace, which is an actual namespace that must be matched.

12. ******What is the difference between Action and ActionSupport ?

Action is interface defines some string like SUCCESS,ERROR etc and an execute() method. For convenience Developer implement this interface to have access to String field in action methods. **ActionSupport** on other hand implements **Action** and some other interfaces and provides some feature like data validation and localized error messaging when extended in the action classes by developers.

13. What is execute and wait interceptor ?

The **Execute** And **Wait** Interceptor is great interceptor provided out of box in Struts2 for running long-lived actions in the background while showing the user a nice progress meter or a progress bar. For example while uploading a large file to the server we can use this interceptor to display a nice running progress bar instead of leaving the user in confusion that the application is not responding. This also prevents the HTTP request from timing out when the action takes more than 5 or 10 minutes.

14. How many different ways can you retrieve the request parameters from within interceptor ?

Two ways

In **the first way** retrieve the HttpServletRequest from the **ActionContext** object and retrieve the parameters the usual way. Code

ActionContext context=(ActionContext)invocation.getInvocationContext();
HttpServletRequest request=(HttpServletRequest)context.get(StrutsStatics.HTTP_REQUEST);
String username=(String)request.getParameter("user");

The second way is pretty much the Struts2, by invoking the **getParameters** method on context object. Code

ActionContext context=(ActionContext)invocation.getInvocationContext();
Map requestParameters=context.getParameters();
String usernames=((String[])m.get("user"))[0];

As you can see the map doesn't return the parameter as String unlike the Servlet specification, but an array of String (can be convenient for multivalued UI controls check box,single value UI controls data can be retrived as in above code).

15. What is abstract package in Struts2, and what is its use ?

An abstract package usually defines inheritable components such as interceptor, different interceptor stacks, result types etc.But doesn't contain any actions. The way we declare a package as abstract is through the package elements attribute as abstract and setting the value to "true". By default (if abstract attribute is not mentioned it is false). Struts-default.xml is an example of abstract package.

Comparing Struts 1 and 2

Feature	**Struts 1**	**Struts 2**
Action classes	Struts 1 requires Action classes to extend an abstract base class. A common problem in Struts 1 is programming to abstract classes instead of interfaces.	An Struts 2 Action *may* implement an Action interface, along with other interfaces to enable optional and custom services. Struts 2 provides a base ActionSupport class to implement commonly used interfaces. Albeit, the Action interface is **not** required. Any POJO object with a execute signature can be used as an Struts 2 Action object.
Threading Model******	Struts 1 Actions are singletons and must be thread-safe since there will only be one instance of a class to handle all requests for that Action. The singleton strategy places restrictions on what can be done with Struts 1 Actions and requires extra care to develop. Action resources must be thread-safe or synchronized.	Struts 2 Action objects are instantiated for each request, so there are no thread-safety issues. (In practice, servlet containers generate many throw-away objects per request, and one more object does not impose a performance penalty or impact garbage collection.)就是说他是 thread-safe 的
Servlet Dependency	Struts 1 Actions have dependencies on the servlet API since the HttpServletRequest and HttpServletResponse is passed to the execute method when an Action is invoked.	Struts 2 Actions are not coupled to a container. Most often the servlet contexts are represented as **simple Maps**, allowing Actions to **be tested in isolation**. Struts 2 Actions can still access the original request and response, if required. However, other architectural elements reduce or eliminate the need to access the HttpServletRequest or HttpServletResponse directly.
Testability****	A major hurdle to testing Struts 1 Actions is that the executemethod	Struts 2 Actions **can be tested by instantiating the Action,** setting

	exposes the Servlet API. A third-party extension, Struts TestCase, offers a set of mock object for Struts 1.	properties, and invoking methods. **Dependency Injection support** also makes testing simpler.
Harvesting Input	Struts 1 uses an ActionForm object to capture input. Like Actions, all ActionForms must extend a base class. Since other JavaBeans cannot be used as ActionForms, developers often create redundant classes to capture input. DynaBeans can used as an alternative to creating conventional ActionForm classes, but, here too, developers may be redescribing existing JavaBeans.	Struts 2 **uses Action properties as input properties**, eliminating the need for a second input object. Input properties may be rich object types which may have their own properties. The Action properties can be accessed from the web page via the taglibs. Struts 2 also **supports the ActionForm pattern**, as well as POJO form objects and POJO Actions. Rich object types, including business or domain objects, can be used as input/output objects. The ModelDriven feature simplifies taglb references to POJO input objects.
Expression Language******	Struts 1 integrates with JSTL, so it uses the JSTL EL. The EL has basic object graph traversal, but relatively weak collection and indexed property support.	Struts 2 can use JSTL, but the framework also supports a more powerful and flexible expression language called "**Object Graph Notation Language" (OGNL).**
Binding values into views	Struts 1 uses the standard JSP mechanism for binding objects into the page context for access.	Struts 2 uses a "**ValueStack**" technology so that the taglibs can access values without coupling your view to the object type it is rendering. The ValueStack strategy（战略） allows reuse of views across a range of types which may have the same property name but different property types.
Type Conversion（转换）	Struts 1 ActionForm properties are usually all Strings. Struts 1 uses Commons-Beanutils for type conversion. Converters are per-class, and not configurable per instance.	Struts 2 uses **OGNL** for type conversion. The framework includes converters for basic and common object types and primitives.
Validation****	Struts 1 supports manual validation via a validate method on the ActionForm, or through an extension to the Commons	Struts 2 supports manual validation via the **validate method** and the **XWork Validation framework.** The Xwork

	Validator. Classes can have different validation contexts for the same class, but cannot chain to validations on sub-objects.	Validation Framework supports chaining validation into sub-properties using the validations defined for the properties class type and the validation context.
Control Of Action Execution	Struts 1 supports separate Request Processors (lifecycles) for each module, but all the Actions in the module must share the same lifecycle.	Struts 2 supports **creating different lifecycles** on a per Action basis via Interceptor Stacks. Custom stacks can be created and used with different Actions, as needed.

16. ******How to create an action with Struts2?

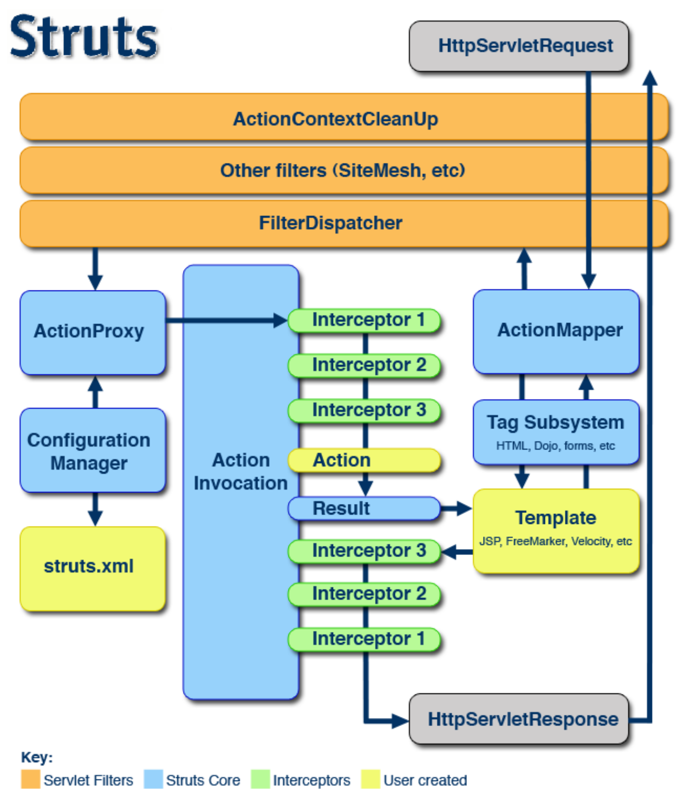
Ans: Creating an action in Struts2 is very different from Struts1 in the sense that there is no mandatory requirement to extend a class from the struts library. A Java bean/POJO which has the private member variables and public getters and setters is sufficient to become a struts action class. As far as execute() method of Struts1 is concerned, a method with return type of String is sufficient. The method name is to be specified in the struts.xml. This change is done so that the testing of struts based application can be done easily.

17. ******What is the advantage of having a POJO class as an action?

Ans: As mentioned in the first question, the POJO class is **light weigh**t and **easy to test** with frameworks like Junit. **Moreover**, there is **no need to create separate ActionForms** to hold the values from the source web page.

18. ******What is the advantage of extending the action class from ActionSupport class?

Ans: The use of ActionSupport class provides the features like validation, locale support and serialization to an action,



19. ******How can one integrate Spring IoC with Struts 2?

Ans: Struts 2 comes with support for Spring and Spring can be used to inject beans to various classes. It can also be used to inject properties to the action class of struts 2. For configuring Spring, **contextLoaderListener** can be configured.就是spsring 用 ioc 注入 struts

20. Describe the flow of a request in a Struts 2 web application?

In the diagram, an initial request goes to the **Servlet container** (such as Jetty or Resin) which is passed through a standard filter chain. The chain includes the (optional)

ActionContextCleanUp filter, which is useful when integrating technologies such as SiteMesh Plugin. Next, the required **FilterDispatcher** is called, which in turn consults the ActionMapper to determine if the request should invoke an action.
If the **ActionMapper** determines that an Action should be invoked, the **FilterDispatcher** delegates（代表） control to the **ActionProxy**. The **ActionProxy** consults the framework **Configuration Files manager** (initialized from the **struts.xml** file). Next, the **ActionProxy** creates an **ActionInvocation**, which is responsible for the command pattern implementation. This includes invoking any **Interceptors** (the *before* clause) in advance of invoking the **Action** itself.
Once the Action returns, the **ActionInvocation** is responsible for looking up the proper **result** associated with the **Action result code** mapped in struts.xml. The result is then executed, which often (but not always, as is the case for **Action Chaining**) involves a template written in JSP or FreeMarker to be rendered. While rendering, the templates can use the Struts Tags provided by the framework. Some of those components will work with the **ActionMapper** to render proper URLs for additional requests.
The filter chain includes:
1) Action ContextCleanUp filter: The ActionContextCleanUp filter is optional and it is useful when integration has to be done with other technologies like SiteMash Plugin.
2） FilterDispatcher: Next the FilterDispatch is called, which in turn uses the ActionMapper to determine weather to invoke an Action. If the action is required to be invoked, the FilterDispatcher delegates the control to the ActionProxy.
3）ActionProxy: The ActionProxy takes the help from Configuration Files manager, which is initialized from the struts.xml. Then the ActionProxy creates an ActionInvocation, which implements the command pattern. The ActionInvocation process invokes the Interceptors (if configured) and then invokes the action. The ActionInvocation looks for proper result. Then the result is executed, which involves the rendering of JSP or templates.

21. What is ActionMapping?

Action mapping contains all the deployment information for a particular Action bean. This class is to determine where the results of the Action will be sent once its processing is complete.
Action mapping contains all the deployment information for a particular Action bean. This class is to **determine** where the results of the Action will be sent once its processing is complete. We can specify the action **mapping** in the configuration file called struts-config.xml. Struts framework creates ActionMapping object from <ActionMapping> configuration element of struts-config.xml file

22. What tool/IDE/frameworks do you use to write code in a Struts 2 web application?

Ans: Mostly it is MyEclipse 9 which has excellent support for struts 2. Netbeans 7 has support for Struts 1 but not Struts 2. Other that the **IDE** one can also mention tools like DreamWeaver, Spring, Hibernate, XMLSpy etc.
******What are the steps to migrate a web application written with Struts 1 to Struts 2?
Ans: This involves **moving** ActionForms of Struts1 to POJO properties of Struts 2. Converting Struts1 struts-config.xml to struts.xml of Struts2.   Rewriting the Struts action classes for Struts 2.

23. ******The interceptors has the following life cycle.

1. Interceptor Object creation
2. initialization

3. intercept
4. destroy

24. What is ActionServlet?

ActionServlet is a simple servlet which is the **backbone** of all Struts applications. It is the **main Controller** component that **handles** client **requests** and **determines** which Action will **process** each received request. It serves as an Action factory – creating specific Action classes based on user's request.

25. What is role of ActionServlet?

ActionServlet performs the role of Controller:
Process user requests
Determine what the user is trying to achieve according to the request
Pull data from the model (if necessary) to be given to the appropriate view,
Select the proper view to respond to the user
Delegates most of this grunt work to Action classes
Is responsible for **initialization** and clean-up of resources

26. Describe validate() and reset() methods ?

validate() : Used to validate properties **after** they have been populated; Called **before** FormBean is handed to Action. Returns a collection of ActionError as ActionErrors. Following is the method signature for the validate() method. publicActionErrors validate(ActionMappingmapping,HttpServletRequestrequest)
reset(): reset() method is called by Struts Framework with each request that uses the defined ActionForm. The purpose of this method is to reset all of the ActionForm's data members prior to the new request values being set. public void reset() {}

27. What are the different kinds of actions in Struts?

The different kinds of actions in Struts are:
ForwardAction
IncludeAction
DispatchAction
LookupDispatchAction
SwitchAction

28. What is DispatchAction?

The **DispatchAction** class is used to **group** related actions into one class. Using this class, you can have a method for each logical action compared than a single execute method. The **DispatchAction** dispatches to one of the logical actions represented by the methods. It picks a method to invoke based on an incoming request parameter. The value of the incoming parameter is the name of the method that the DispatchAction will invoke.

29. How to use DispatchAction?

To use the **DispatchAction**, follow these steps :
Create a class that extends DispatchAction (instead of Action)

In a new class, **add** a method for every function you need to perform on the service – The method has the same signature as the execute() method of an Action class.
Do not override execute() method – Because DispatchAction class itself provides execute() method.
Add an entry to struts-config.xml

30. What is ForwardAction ?

Struts provides a **built-in Action** class called **ForwardAction** to avoid directly accessing the JSP. With **ForwardAction**, the Struts Controller is still in the loop while navigating from PageA to PageB.

31. What is the use of ForwardAction?

The **ForwardAction** class is useful when you're trying to integrate Struts into an existing application that uses Servlets to perform business logic functions. You can use this class to take advantage of the Struts controller and its functionality, without having to rewrite the existing Servlets. Use **ForwardAction** to forward a request to another resource in your application, such as a Servlet that already does business logic processing or even another JSP page. By using this predefined action, you don't have to write your own Action class. You just have to set up the struts-config file properly to use ForwardAction.

32. What is IncludeAction?

The **IncludeAction** class is useful when you want to **integrate** Struts into an **application** that uses **Servlets**. Use the **IncludeAction** class to include another resource in the response to the request being processed.
What is the difference between ForwardAction and IncludeAction?
The difference is that you need to use the **IncludeAction** only if the action is going to be included by another action or jsp. Use **ForwardAction** to forward a request to another resource in your application, such as a Servlet that already does business logic processing or even another JSP page.

33. What is LookupDispatchAction?

The **LookupDispatchAction** is a subclass of DispatchAction. It does a reverse lookup on the resource bundle to get the key and then gets the method whose name is associated with the key into the Resource Bundle.

34. What is the use of LookupDispatchAction?

LookupDispatchAction is useful if the method name in the Action is **not driven** by its **name** in the **front end**, but **by** the **Locale independent key** into the resource bundle. Since the key is always the same, the **LookupDispatchAction** shields your application from the side effects of I18N.

35. What is difference between LookupDispatchAction and DispatchAction?

The difference between **LookupDispatchAction** and **DispatchAction** is that the actual method that gets called in **LookupDispatchAction** is based on a **lookup of a key value** instead of specifying the method name directly.

36. What is SwitchAction?

The **SwitchAction** class provides a means to switch from a **resource** in one module to another resource in a different module. **SwitchAction** is useful only if you have multiple modules in your Struts application. The SwitchAction class can be used as is, without extending.

37. What is DynaActionForm?

A specialized **subclass** of **ActionForm** that allows the creation of form beans with **dynamic sets** of **properties** (configured in configuration file), without requiring the developer to create a Java class for each type of form bean.

38. What is difference between ActionForm and DynaActionForm?

An **ActionForm** represents an HTML form that the user interacts with over one or more pages. You will provide properties to hold the state of the form with getters and setters to access them. Whereas, using **DynaActionForm** there is no need of providing properties to hold the state. Instead these properties and their type are declared in the struts-config.xml
The **DynaActionForm** bloats up the Struts config file with the xml based definition. This gets annoying as the Struts Config file grow larger.
The **DynaActionForm** is not strongly typed as the **ActionForm**. This means there is no compile time checking for the form fields. Detecting them at runtime is painful and makes you go through redeployment.
ActionForm can be cleanly organized in packages as against the flat organization in the Struts Config file.
ActionForm were designed to act as a Firewall between HTTP and the Action classes, i.e. isolate and encapsulate the HTTP request parameters from direct use in Actions. With **DynaActionForm**, the property access is no different than using request.getParameter(..).
DynaActionForm construction at runtime requires a lot of Java Reflection (Introspection) machinery that can be avoided.

八、 Spring Interview Questions

1. ******What is IOC (or Dependency Injection)?

The basic concept of the **Inversion of Control** pattern (also known as **dependency injection**) is that you do not create your objects but **describe** how they should be created. You **don't directly connect** your components and services together in code but **describe** which services are needed by which components in a configuration file. A **container** (in the case of the Spring framework, the IOC container) is then **responsible** for hooking it all up.（i.e., Applying IoC, objects are given their dependencies at creation time by some external entity that coordinates each object in the system. That is, dependencies are injected into objects.） So, **IoC** means an inversion of responsibility with regard to how an object obtains references to collaborating objects.

2. ******What are the different types of IOC (dependency injection) ?

There are **2 types** of dependency injection spring support:
Constructor Injection (e.g. Pico container, Spring etc): Dependencies are provided as constructor parameters.
Setter Injection (e.g. Spring): Dependencies are assigned through JavaBeans properties (ex: setter methods).
Note: Spring supports only Constructor and Setter Injection

3. ******What are the benefits of IOC (Dependency Injection)? 就是 IOC 的优点

Benefits of IOC (Dependency Injection) are as follows:

- **Minimizes the amount of code:** in your application. With IOC containers you do not care about how services are created and how you get references to the ones you need. You can also easily add additional services by adding a new constructor or a setter method with little or no extra configuration.
- **Make your application more testable:** by not requiring any singletons or JNDI lookup mechanisms in your unit test cases. IOC containers make unit testing and switching implementations very easy by manually allowing you to inject your own objects into the object under test.
- IOC containers **support eager instantiation and lazy loading** of services: Containers also provide support for instantiation of managed objects, cyclical dependencies, life cycles management, and dependency resolution between managed objects etc.
- **Loose coupling:** Components can add declaratively so we can add and remove the components without code change.
- **Not Required any singletons**: Don't need to code for singleton class. Every class is by **default singleton**. you can make not singleton by making **singleton="false"**
- **No** App Server **Dependent** ? like EJB JNDI Calls

4. ********What is Spring ?**

Spring is an open source framework created to address the complexity of enterprise application development. One of the **chief advantages** of the Spring framework is its layered architecture, which allows you to be selective about which of its components you use while also providing a cohesive framework for J2EE application development.

5. ********What are the advantages of Spring framework?** ***Srping 的优点***

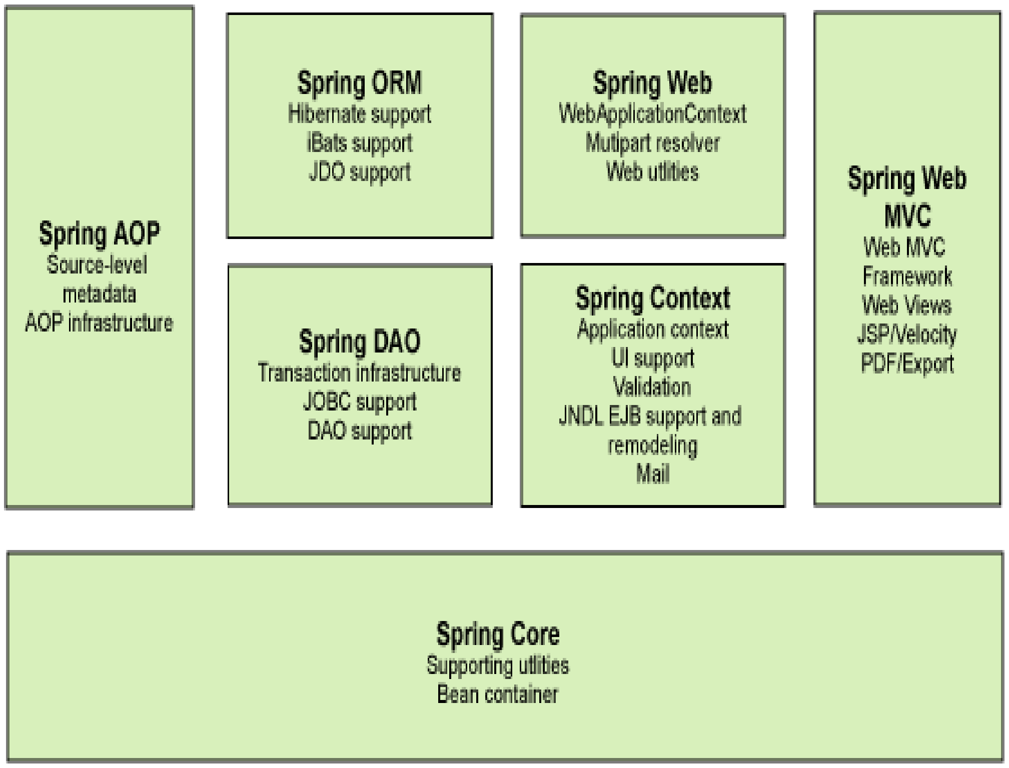
The advantages of Spring are as follows:

- Spring has **layered architecture**. Use what you need and leave you don't need now.
- Spring Enables **POJO Programming.** There is no behind the scene magic here. POJO programming enables continuous integration and testability.
- Dependency Injection and Inversion of Control **Simplifies JDBC**
- **Open source** and no vendor lock-in.
- **Lightweight** container
- ***No* App** Server Dependent. like EJB JNDI Calls
- Objects are created **Lazily** , **Singleton** - configuration
- Components can **added Declaratively Initialization** of properties is **easy** ? no need to read from properties file
- Declarative transaction, security and logging service – **AOP** application code is much **easier** to **unit test**
- With a **Dependency Injection** approach, dependencies are explicit, and evident in constructor or JavaBean properties
- Spring's configuration management services can be used **in** any **architectural layer**, in whatever runtime environment.
- Spring can effectively **organize** your **middle tier** objects
- not requires **special deployment steps**

6. ********How many modules are there in Spring? What are they?**

Spring comprises of seven modules. They are..

The core container:
The core container provides the essential functionality of the Spring framework. A primary component of the core container is the BeanFactory, an implementation of the Factory pattern. The BeanFactory applies the Inversion of Control (IOC) pattern to separate an application's configuration and dependency specification from the actual application code.



Spring context:
The Spring context is a configuration file that provides context information to the Spring framework. The Spring context includes enterprise services such as JNDI, EJB, e-mail, internalization, validation, and scheduling functionality.

Spring AOP:
The Spring AOP module integrates aspect-oriented programming functionality directly into the Spring framework, through its configuration management feature. As a result you can easily AOP-enable any object managed by the Spring framework. The Spring AOP module provides transaction management services for objects in any Spring-based application. With Spring AOP you can incorporate declarative transaction management into your applications without relying on EJB components.

Spring DAO:
The Spring **JDBC DAO abstraction layer** offers a meaningful exception hierarchy for managing the exception handling and error messages thrown by different database vendors. The exception hierarchy simplifies error handling and greatly reduces the amount of exception code you need to write, such as opening and closing connections. Spring DAO's JDBC-oriented exceptions comply to its generic DAO exception hierarchy.

Spring ORM:
The Spring framework plugs into several ORM(Object relational mapping) frameworks to provide its Object Relational tool, including JDO, Hibernate, and iBatis SQL Maps. All of these comply to Spring's generic transaction and DAO exception hierarchies.

Spring Web module:

The Web context module builds on top of the application context module, providing contexts for Web-based applications. As a result, the Spring framework supports integration with Jakarta Struts. The Web module also eases the tasks of handling multi-part requests and binding request parameters to domain objects.

Spring MVC framework:

The Model-View-Controller (MVC) framework is a full-featured MVC implementation for building Web applications. The MVC framework is highly configurable via strategy interfaces and accommodates numerous view technologies including JSP, Velocity, Tiles, iText, and POI.

7. ******What is Bean Factory ?

A **BeanFactory** is like a factory class that contains a collection of beans. The **BeanFactory** holds Bean Definitions of multiple beans within itself and then instantiates the bean whenever asked for by clients.

BeanFactory is able to create associations between collaborating objects as they are instantiated. This removes the burden（负担） of configuration from bean itself and the beans client.

BeanFactory also takes part in the life cycle of a bean, making calls to custom initialization and destruction methods.

8. ******What is the typical Bean life cycle in Spring Bean Factory Container ?就是 spring 的生命周期

Bean life cycle in Spring Bean Factory Container is as follows:

The spring container **finds** the bean's **definition** from the **XML** file and **instantiates** the bean.

Using the **dependency injection**, spring populates all of the properties as **specified** in the bean **definition**

If the bean implements the **BeanNameAware** interface, the factory calls **setBeanName()** parsing the bean's ID.

If the bean implements the **BeanFactoryAware** interface, the factory calls **setBeanFactory()**, passing an instance of itself.

If there are any **BeanPostProcessors** associated with the bean, their **post- ProcessBeforeInitialization**() methods will be called.

If an **init-method** is specified for the bean, it will be called.

Finally, **if** there are any **BeanPostProcessors** associated with the bean, their **postProcessAfterInitialization**() methods will be called.

What is ApplicationContext?

an **applicationcontext** is **same** as a bean factory. **Both** load bean definitions, wire beans together, and dispense（免除） beans upon request. But it also provides:

A means for resolving text messages, including support for internationalization.

A generic way to load file resources.

Events to beans that are registered as listeners.

9. ******What is the difference between Bean Factory and ApplicationContext ?

On the surface, an **application context** is same as a **bean factory**. But **application context** offers much more..

Application contexts provide a means for resolving(分解) text messages, including support for internationalization of those messages.

Application contexts provide a generic way to **load file resources**, such as images.

Application contexts can **publish** events to **beans** that are registered as **listeners**.
Certain operations on the **container** or beans in the **container**, which have to be handled in a programmatic fashion with a **bean factory,** can be handled declaratively in an **application context**.
ResourceLoader support: Spring's Resource interface is a flexible generic abstraction for handling low-level resources. **An application context** itself is a **ResourceLoader**, Hence provides an application with access to deployment-specific Resource instances.
MessageSource support: The **application context** implements **MessageSource**, an interface used to obtain localized messages, with the actual implementation being pluggable

10. ******What are the common implementations of the ApplicationContext ?

The **three commonly** used implementation of '**ApplicationContext'** are
ClassPathXmlApplicationContext : It Loads context definition from an XML file located in the classpath, treating context definitions as classpath resources. The application context is loaded from the application's classpath by using the code .ApplicationContext context = new ClassPathXmlApplicationContext("bean.xml");
FileSystemXmlApplicationContext : It loads context definition from an XML file in the filesystem. The application context is loaded from the file system by using the code .ApplicationContext context = new FileSystemXmlApplicationContext("bean.xml");
XmlWebApplicationContext :It loads context definition from an XML file contained within a web application.
How is a typical spring implementation look like ? spring 的一些成员
For a typical Spring Application we need the following files:
An **interface** that defines the functions.
An **Implementation** that contains properties, its setter and getter methods, functions etc.,
Spring **AOP** (Aspect Oriented Programming)
A **XML file** called Spring configuration file.
Client program that uses the function.

11. ******How to integrate your Struts application with Spring?

To integrate your Struts application with Spring, we have **two options:**
Configure Spring to manage your Actions as beans, using the ContextLoaderPlugin, and set their dependencies in a Spring context file.
Subclass Spring's ActionSupport classes and grab your Spring-managed beans explicitly using a getWebApplicationContext() method.

12. ******What do you mean by Bean wiring ?

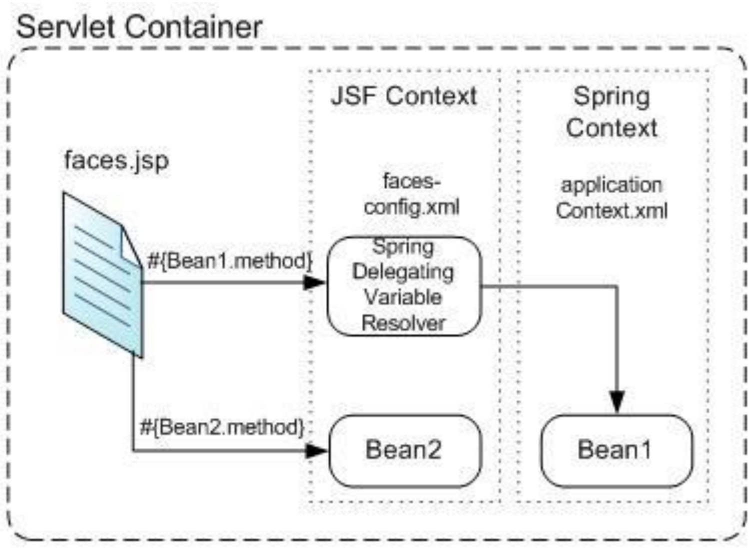
The act of creating **associations**(关联) between **application components** (beans) within the **Spring container** is referred to(被叫做) as Bean wiring.

13. ******What do you mean by AutoWiring?

The Spring container is able to **autowire** relationships **between** collaborating beans. This means that it is possible to automatically let Spring resolve collaborators (other beans) for your bean by inspecting the contents of the BeanFactory. The autowiring functionality has five modes. no , byname, byType, constructor, autodirect

What is DelegatingVariableResolver?
Spring provides a custom JavaServer Faces VariableResolver implementation that extends the standard Java Server Faces managed beans mechanism which lets you use **JSF** and **Spring** together. This variable resolver is called as DelegatingVariableResolver

14. How to integrate Java Server Faces (JSF) with Spring?
JSF and Spring do share some of the same features, most noticeably in the area of IOC services. By **declaring** JSF managed-beans in the faces-config.xml configuration file, you allow the FacesServlet to instantiate that bean at startup. Your JSF pages have access to these beans and all of their properties.We can integrate JSF and Spring in **two ways:**
DelegatingVariableResolver: Spring comes with a JSF variable resolver that lets you use JSF and



Spring together.
The DelegatingVariableResolver will first delegate value lookups to the default resolver of the underlying JSF implementation, and then to Spring's 'business context' WebApplicationContext. This allows one to easily inject dependencies into one's JSF-managed beans.
FacesContextUtils:customVariableResolver works well when mapping one's properties to beans in faces-config.xml, but at times one may need to grab a bean explicitly. The FacesContextUtils class makes this easy. It is similar to WebApplicationContextUtils, except that it takes a FacesContext parameter rather than a ServletContext parameter.
ApplicationContextctx =
FacesContextUtils.getWebApplicationContext(FacesContext.getCurrentInstance());

15. What are ORM’s Spring supports ? orm 其实就是面向对象数据库
Spring supports the following **ORM’s :**
Hibernate ， iBatis, JPA (Java Persistence API) , TopLink, JDO (Java Data Objects) ,OJB
******What are the ways to access Hibernate using Spring ?
There are **two approaches** to Spring’s Hibernate integration:
Inversion of Control with a HibernateTemplate and Callback (ioc)
Extending **HibernateDaoSupport** and Applying an AOP Interceptor(dao)

16. ******How to integrate Spring and Hibernate using HibernateDaoSupport?

Spring and Hibernate can integrate using Spring's **SessionFactory** called **LocalSessionFactory**. The integration process is of **3 steps.**

Configure the Hibernate **SessionFactory**

Extend your **DAO** Implementation from **HibernateDaoSupport**

Wire in **Transaction** Support with **AOP**

17. What are Bean scopes in Spring Framework ?

If you are using ApplicationContext then five scopes (singleton, prototype, request, session, globalsession) and if you are using BeanFactory then two(singleton, prototype)

Scope	Description
singleton	Scopes a single bean definition to **a single object instance** per Spring IoC container.
prototype	Scopes a single bean definition to **any number of object instances**.
request	Scopes a single bean definition to the lifecycle of **a single HTTP request;** that is each and every HTTP request will have its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring ApplicationContext.
session	Scopes a single bean definition to the lifecycle of **a HTTP Session**. Only valid in the context of a web-aware Spring ApplicationContext.
global session	Scopes a single bean definition to the lifecycle of **a global HTTP Session**. Typically only valid when used in a portlet context. Only valid in the context of a web-aware Spring ApplicationContext.

18. ******What is AOP?

Aspect-oriented programming, or **AOP**, is a programming technique that allows programmers to **modularize crosscutting concerns**, or behavior that **cuts across** the typical **divisions** of **responsibility**, such as logging and transaction management. The core construct of AOP is the aspect, which encapsulates behaviors affecting multiple classes into reusable modules. (crosscutting the functions and set the cut point.)

19. ******How the AOP used in Spring?

AOP is used in the Spring Framework: To provide declarative enterprise services, especially as a replacement for EJB declarative services. The **most important** such service is **declarative transaction management**, which builds on the Spring Framework's transaction abstraction. To allow users to implement custom aspects, complementing their use of OOP with AOP.

20. ******What do you mean by JointPoint?

A **point** during the execution of a program, such as the execution of a method or the handling of an exception. In Spring AOP, a **join point** always represents a method execution.

21. ******What do you mean by Advice?

Action taken by an aspect at a particular join point. **Different types** of **advice** include **"around**," "**before**" and "**after**" advice. **Many AOP frameworks,** including Spring, **model** an advice **as** an **interceptor**, maintaining a chain of interceptors "around" the join point.

22. ******What are the types of Advice?

Types of advice:
Before advice: Advice that executes before a join point, but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception).
After returning advice: Advice to be executed after a join point completes normally: for example, if a method returns without throwing an exception.
After throwing advice: Advice to be executed if a method exits by throwing an exception.
After (finally) advice: Advice to be executed regardless of the means by which a join point exits (normal or exceptional return).
Around advice: Advice that surrounds a join point such as a method invocation. This is the most powerful kind of advice. Around advice can perform custom behavior before and after the method invocation. It is also responsible for choosing whether to proceed to the join point or to shortcut the advised method execution by returning its own return value or throwing an exception
What are the types of the transaction management Spring supports ?
Spring Framework supports:
Programmatic transaction management.
Declarative transaction management.

23. ******What are the benefits of the Spring Framework transaction management ? 事物管理的好处

The Spring Framework provides a consistent abstraction for transaction management that delivers the following benefits:
Provides a consistent programming model **across different transaction APIs** such as JTA, JDBC, Hibernate, JPA, and JDO.
Supports **declarative transaction** management.
Provides a **simpler API** for programmatic transaction management than a number of complex transaction APIs such as JTA.
Integrates very well with Spring's various data access abstractions.
Why most users of the Spring Framework choose declarative transaction management ?
Most users of the Spring Framework choose **declarative transaction management** because it is the option with the least impact on application code, and hence is most consistent with the ideals of a non-invasive(非侵略) lightweight container.

24. Explain the similarities and differences between EJB CMT and the Spring Framework's declarative transaction management ?

The basic approach is **similar**: it is possible to specify transaction behavior (or lack of it) down to individual method level. It is possible to make a setRollbackOnly() call within a transaction context if necessary.
The **differences** are:

Unlike **EJB CMT,** which is tied to JTA, the **Spring Framework's** declarative transaction management works in any environment. It can work with JDBC, JDO, Hibernate or other transactions under the covers, with configuration changes only.

The Spring Framework enables declarative transaction management to be applied to any class, not merely special classes such as EJBs.

The Spring Framework offers declarative rollback rules: this is a feature with no EJB equivalent. Both programmatic and declarative support for rollback rules is provided.

The Spring Framework gives you an opportunity to customize transactional behavior, using AOP. With **EJB CMT,** you have no way to influence the container's transaction management other than setRollbackOnly().

The **Spring Framework** does **not** support propagation of transaction contexts across **remote calls**, as do high-end application servers.

When to use programmatic and declarative transaction management ?

Programmatic transaction management is usually a good idea only if you have a small number of transactional operations. On the other hand, if your application has numerous transactional operations, **declarative transaction management** is usually worthwhile. It keeps transaction management out of business logic, and is not difficult to configure.

25. ******Explain about the Spring DAO support ?

The **Data Access Object (DAO)**. This allows one to **switch** between the persistence technologies fairly **easily** and it also allows one to code **without** worrying about catching exceptions that are specific to each technology.

26. What is Spring's JdbcTemplate ?

Spring's **JdbcTemplate** is central class to interact with a database through JDBC. **JdbcTemplate** provides many convenience methods for doing things such as converting database data into primitives or objects, executing prepared and callable statements, and providing custom database error handling.

JdbcTemplate template = new JdbcTemplate(myDataSource);

27. What is PreparedStatementCreator ?

PreparedStatementCreator:

Is one of the most common used interfaces for **writing data** to **database**; Has one method – createPreparedStatement(Connection)

Responsible for creating a PreparedStatement ; Does **not** need to **handle SQLExceptions**.

28. What are the differences between EJB and Spring ?

Spring and EJB feature comparison.

Feature	EJB	Spring
Transaction management	Must use a JTA transaction manager. Supports transactions that span remote method calls.	Supports multiple transaction environments through its PlatformTransactionManager interface, including JTA, Hibernate, JDO, and JDBC. Does not natively support distributed

		transactions—it must be used with a JTA transaction manager.
Declarative transaction support	Can define transactions declaratively through the deployment descriptor. Can define transaction behavior per method or per class by using the wildcard character *. Cannot declaratively define rollback behavior—this must be done programmatically.	Can define transactions declaratively through the Spring configuration file or through class metadata. Can define which methods to apply transaction behavior explicitly or by using regular expressions. Can declaratively define rollback behavior per method and per exception type.
Persistence	Supports programmatic bean-managed persistence and declarative container managed persistence.	Provides a framework for integrating with several persistence technologies, including JDBC, Hibernate, JDO, and iBATIS.
Declarative security	Supports declarative security through users and roles. The management and implementation of users and roles is container specific. Declarative security is configured in the deployment descriptor.	No security implementation out-of-the box. Acegi, an open source security framework built on top of Spring, provides declarative security through the Spring configuration file or class metadata.
Distributed computing	Provides container-managed remote method calls.	Provides proxying for remote calls via RMI, JAX-RPC, and web services.

九、 Database Interview Questions & Answers

1. ******What is JDBC?

JDBC is a **layer** of abstraction that allows users to choose between databases. JDBC allows you to write database applications in Java without having to concern yourself with the underlying details of a particular database. JDBC API has **two major packages** java.sql and javax.sql.
There are **4 types** of **JDBC drivers** available----

2. Type 1 Driver- the JDBC-ODBC bridge :

is a database driver implementation that the ODBC driver to connect to the database. The driver converts JDBC method calls into ODBC function calls. The bridge is usually used when there is no pure-Java driver available for a particular database. The driver is implemented in the **sun.jdbc.odbc.JdbcOdbcDriver** class .The driver is **platform-dependent** as it makes use of ODBC which in turn depends on native libraries of the operating system.

Advantage :

Almost **any database** for which ODBC driver is installed, can be accessed.

Disadvantage :

a) Performance **overhead** since the calls have to go through the JDBC overhead bridge to the ODBC driver.

b) The ODBC driver needs to be **installed** on the **client** machine

c) considering the **client-side** software needed, this might not be suitable for applets.

3. Type 2 Driver - the Native-API Driver :

is a database driver implementation that uses the **client-side libraries** of the database. The driver **converts** JDBC method calls **into** native calls of the database API. The type 2 driver is not written entirely in Java as it interfaces with non-Java code that makes the final database calls. **A native-API** partly Java technology-enabled driver converts JDBC calls into calls on the client API for **ORACLE**, **DB2** or other . **Note** that, like the bridge driver, this style of driver requires that some binary code be loaded on each client machine. **However** the type 2 driver provides more functionality and performance than the type 1 driver as it does not have the overhead of the additional ODBC function calls.

Advantage:

Better performance than Type 1 since no jdbc to odbc translation is needed

Disadvantage :

a) The vendor client library needs to be **installed** on the **client** machine.

b) **Cannot** be **used** in **internet** due the client side software needed

c) **Not** all databases give the **client** side **library**

4. Type 3 driver - the Network-Protocol Driver:

is a database driver implementation which makes use of a **middle-tier** between the calling program and the database. The middle-tier (application server) **converts** JDBC calls directly or indirectly **into** the vendor-specific database protocol.

Advantages:

a) Since the communication between client and the middleware server is database independent, there is **no need** for the vendor db library **on** the **client machine**.

b) The **Middleware Server** (Can be a full fledged J2EE Application server) can **provide** typical **middleware services** like caching (connections, query results, and so on), load balancing etc.

Disadvantages :

a) **Requires** database-specific **coding** to be done in the middle tier.

b) An **extra layer** added may result in a time-bottleneck

5. Type 4 - the Native-Protocol Driver :

is a database driver implementation that converts JDBC calls directly into the vendor-specific database protocol. The type 4 driver is written completely in Java and is hence platform independent. It is installed inside the Java Virtual Machine of the client. It provides better performance over the type 1 and 2 drivers as it does not have the overhead of conversion of calls into ODBC or database API calls. Unlike the type 1 and 2 drivers, it does not need associated software to work..

Advantages :

a) These drivers don't translate the requests into db request to ODBC or pass it to client api for the db, nor do they need a middleware layer for request indirection. Thus the **performance** is considerably **improved**.
b) **Web application** mainly **used** this driver.
Disadvantage:
At **client side**, a **separate driver** is needed for each database. ex- classes12.zip (for ORACLE)

6. ******What are the steps in the JDBC connection?
While making a JDBC connection we go through the following steps :
Step 1 : Register the database **driver** by using : Class.forName("driver classs for that specific database\");
Step 2 : Now **create** a database **connection** using :
Connection con = DriverManager.getConnection(url,username,password);
Step 3: Now **Create** a **query** using : Statement stmt = Connection.Statement(\"select * from EMP\");
Step 4 : Exceute the **query** : ResultSet rs = stmt.exceuteQuery();

7. What is Rowset ? CachedRowSet, JDBCRowSet and WebRowSet ?
A **RowSet** object is a java bean component and **extends** the **ResultSet interface**. The RowSet is majorly classified into **two types** :
Connected Rowset :
The **connected rowset** as the name suggests in connected to the database connection object like the resultset. The **JDBCRowSet** is the **example** of the connected RowSet.
2) Disconnected RowSet:
The **disconnected RowSet** only connects to the database **whenever required** and **after finishing** the interaction they **close** the database connection
There are three types of RowSet :
A CachedRowSet class: a disconnected rowset that caches its data in memory; not suitable for very large data sets, but an ideal way to provide thin Java clients.
A JDBCRowSet class: a connected rowset that serves mainly as a thin wrapper around a ResultSet object to make a JDBC driver look like a JavaBeans component.
A WebRowSet class: a connected rowset that uses the HTTP protocol internally to talk to a Java servlet that provides data access; used to make it possible for thin web clients to retrieve and possibly update a set of rows.

8. ******what the difference between rowset and resultset ?
Both **rowset** and **resultset** is contains result data from executed query. But after connection close **resultset** data will be lost. But in case of **rowset** you can get the data after connection close also.
What is Batch Updates Using Statements in JDBC ?
Batch Updates calls to database as a chunk. If you want to run **more than one sql** statement **in a single database call** then you have to go for **Batch Update**. All Insert in single data base call and single transaction (if one fail the all fail)就是将多个语句放在一起执行
Statement stmt = con.createStatement();
    stmt.addBatch("INSERT INTO EMP VALUES(1,'Ram1')");
    stmt.addBatch("INSERT INTO EMP VALUES(2,'Ram2')");

stmt.addBatch("INSERT INTO EMP VALUES(3,'Ram3')");

9. ******Handling Blob and CLOB data using JDBC ? Insert CLOB and Retrive CLOB , Convert into String ?

BLOB (Binary Large Objects) and **CLOB**(Character large objects) are special datatypes and can hold the large chunks of data in form of objects or text. Blob and Clob objects persist the data of the objects into the database as a **stream**

10. ******What is PreparedStatement ?

PreparedStatement is precompiled statement. Fist time it compile the SQL query and next **call** it **only** pass the parameter value and execute, in the same connection. **PreparedStatement** is **not** compiling query **every time**. **PreparedStatement** is better for CLOB and BLOB object.

11. ******Difference between Statement , PreparedStatement and CallableStatement ? 主要是 preparedstatement 的优点也可以在这里看出

Statement : Statement **every time compile** the SQL and Execute.
PreparedStatement : If we are using **PreparedStatement** the **execution** time will be **less**. First time RDBMS compile the SQL and **PreparedStatement** is executed then other calls doesn't compile the SQL **only execute** the SQL within the connection live. you must use a **PreparedStatement** object if you want to use **large objects** like BLOBs or CLOBs. **PreparedStatement** is its **support** for **batching.** Objects can be reused with passing different values to the queries.
CallableStatement : CallableStatement is for call to a **stored procedure**.

12. ******What do mean by Connection pooling? 还有优点在里面

Opening and closing of database connections is a costly(resource intensive).So a **pool** of database **connections** is obtained **at start** up by the application server and maintained in a pool. When there is a **request** for a **connection** from the application, the application server **gives** the **connection** from the **pool** and when **closed** by the application is **returned** back to the pool. Min and max size of the connection pool is configurable. **优点是:** This technique provides better handling of database connectivity.
J2EE 服务器启动时会建立一定数量的池连接，并一直维持不少于此数目的池连接。客户端程序需要连接时，池驱动程序会返回一个未使用的池连接并将其表记为忙。如果当前没有空闲连接，池驱动程序就新建一定数量的连接，新建连接的数量有配置参数决定。当使用的池连接调用完成后，池驱动程序将此连接表记为空闲，其他调用就可以使用这个连接。

13. ******What are the difference between Stored Procedure and Trigger in SQL Server? 这个曾经被问到过

1). **trigger** is run automatically if the event is occurred but **s.p** don't run automatically but you have to run it manually
2). **SP** can pass the parameters which is not a case with Triggers.
3）.A **Trigger** can call the specific **SP** in it but the **reverse** is not true

存储过程 是执行一组 sql 语句，将一个复杂的操作过程放到一个 sp 里面，由数据库服务器处理，提高运行的效率和保证数据的完整性。
触发器 是一个表数据的变更后通过触发器来修改与之相关联的其他表的数据，保证数据的一致性。

14. What is Metadata and why should I use it?

Metadata is information about one of two things: Database information (java.sql.DatabaseMetaData), or Information about a specific ResultSet (java.sql.ResultSetMetaData). Use **DatabaseMetaData** to find information about your database, such as its capabilities and structure. Use **ResultSetMetaData** to find information about the results of an SQL query, such as size and types of columns

What will Class.forName do while loading drivers?

It is used to **create** an **instance** of a driver and **register** it with the **DriverManager**. When you have loaded a driver, it is available for **making** a **connection** with a **DBMS**

15. ******What is Dirty read?

A has changed a row, but has **not committed** the **changes**. B reads the uncommitted data but his view of the data may be wrong if A **rolls back** his changes and updates his own changes to the database. 就是一个改了数据没有确定，然后回滚了，但是在回滚前数据已经被别人读了，就会出现 dirty read。

What is Single-Phase Commit ?

If **only** a **single resource** (database) is **enlisted** in the **transaction**, you **can use** single-phase commit. 就是直接发送 commit 完事

What is two-phase commit?

Two-phase commit is a **transaction protocol** designed for the complications that arise with **distributed resource managers**. With a **two-phase commit protocol**, the distributed transaction manager employs a coordinator to manage the individual resource managers.是先发送 prepare，然后客户回复 ok，然后再发送 commit；如果客户不回复，就发送 rollback。

什么情况下使用： When a **transaction** involves **multiple distributed resources**, for example, a database server on each of two different network hosts, the commit process is somewhat complex because the transaction includes operations that span two distinct software systems, each with its own resource manager, log records, and so on. (In this case, the distributed resources are the database servers.)

16. ******The five Normal Forms

First Normal Form: 'A relation R is in first normal form (1NF) if and only if all underlying domains contain atomic values only.' 每个项不可再分

Second Normal Form: 'A relation R is in second normal form (2NF) if and only if it is in 1NF and every nonkey attribute is fully dependent on the primary key.' 拥有主键

Third Normal Form : 'A relation R is in third normal form (3NF) if and only if it is in 2NF and every nonkey attribute is nontransitively dependent on the primary key.' 两个表中的非主键项不能再另外一个表里也出现

Boyce/Codd Normal Form : 'A relation R is in Boyce/Codd normal form (BCNF) if and only if every determinant is a candidate key.' 基于第三范式，

第一范式（1NF）： 字段具有原子性,不可再分。所有关系型数据库系统都满足第一范式）

数据库表中的字段都是单一属性的，不可再分。例如，姓名字段，其中的姓和名必须作为一个整体，无法区分哪部分是姓，哪部分是名，如果要区分出姓和名，必须设计成两个独立的字段。

第二范式（2NF）：

第二范式（2NF）是在第一范式（1NF）的基础上建立起来的，即满足第二范式（2NF）必须先满足第一范式（1NF）。要求数据库表中的每个实例或行必须可以被惟一地区分。通常需要为表加上一个列，以存储各个实例的惟一标识。这个惟一属性列被称为主关键字或主键。**第二范式（2NF）**要求实体的属性完全依赖于主关键字。所谓完全依赖是指不能存在仅依赖主关键字一部分的属性，如果存在，那么这个属性和主关键字的这一部分应该分离出来形成一个新的实体，新实体与原实体之间是一对多的关系。为实现区分通常需要为表加上一个列，以存储各个实例的惟一标识。简而言之，第二范式就是非主属性非部分依赖于主关键字。

第三范式的要求如下：

满足第三范式（3NF）必须先满足第二范式（2NF）。简而言之，第三范式（3NF）要求一个数据库表中不包含已在其它表中已包含的非主关键字信息。

所以第三范式具有如下特征：

1，每一列只有一个值

2，每一行都能区分。

3，每一个表都不包含其他表已经包含的非主关键字信息。

例如，帖子表中只能出现发帖人的 id，而不能出现发帖人的 id，还同时出现发帖人姓名，否则，只要出现同一发帖人 id 的所有记录，它们中的姓名部分都必须严格保持一致，这就是数据冗余。

BC 范式

在第三范式的基础上，数据库表中如果不存在任何字段对任一候选关键字段的传递函数依赖则符合第三范式。

假设仓库管理关系表为 StorehouseManage(仓库 ID, 存储物品 ID, 管理员 ID, 数量)，且有一个管理员只在一个仓库工作；一个仓库可以存储多种物品。这个数据库表中存在如下决定关系：

(仓库 ID, 存储物品 ID) →(管理员 ID, 数量)

(管理员 ID, 存储物品 ID) → (仓库 ID, 数量)

所以，(仓库 ID, 存储物品 ID)和(管理员 ID, 存储物品 ID)都是 StorehouseManage 的候选关键字，表中的唯一非关键字段为数量，它是符合第三范式的。但是，由于存在如下决定关系：

(仓库 ID) → (管理员 ID)

(管理员 ID) → (仓库 ID)

即存在关键字段决定关键字段的情况，所以其不符合 BCNF 范式。它会出现如下异常情况：

(1) 删除异常：

当仓库被清空后，所有"存储物品 ID"和"数量"信息被删除的同时，"仓库 ID"和"管理员 ID"信息也被删除了。

(2) 插入异常：

当仓库没有存储任何物品时，无法给仓库分配管理员。

(3) 更新异常：

如果仓库换了管理员，则表中所有行的管理员 ID 都要修改。

把仓库管理关系表分解为二个关系表：

仓库管理：StorehouseManage(仓库 ID, 管理员 ID)；

仓库：Storehouse(仓库 ID, 存储物品 ID, 数量)。

这样的数据库表是符合 BCNF 范式的，消除了删除异常、插入异常和更新异常。

17. ******Optimization table design 数据库优化

用 PreparedStatement

Foreign Key will affect the performance of inserting and delete. Do not use they as possible.
Allow property redundancy. Eg: one message's reply number and time can be existed in several tables.
Using index

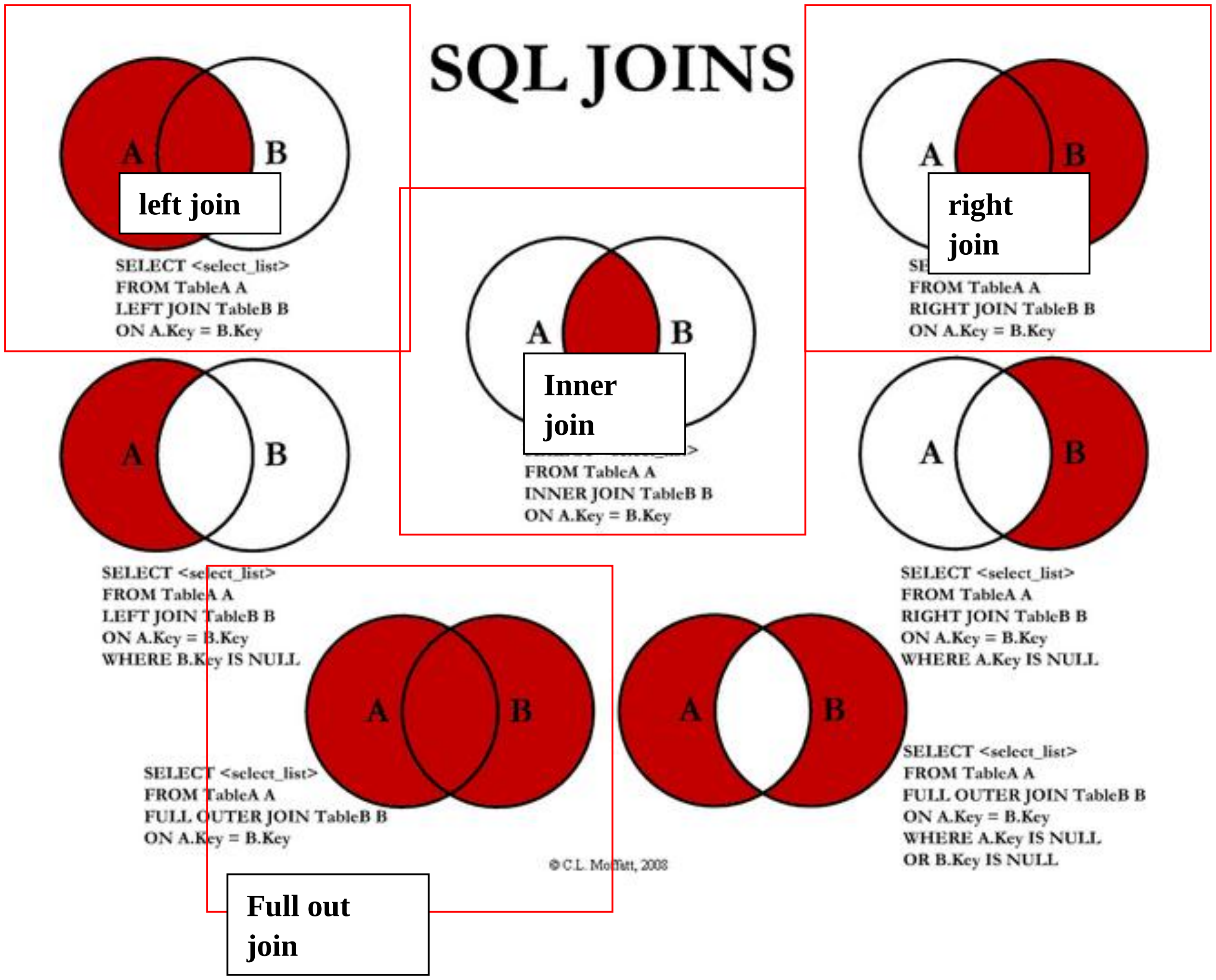
18. ******Union 和 Union All 区别

UNION：The **UNION** command is used to select related information from two tables, much like the JOIN command. However, when using the UNION command all selected columns need to be of the same data type. With UNION, only distinct values are selected.在进行表链接后会筛选掉重复的记录，所以在表链接后会对所产生的结果集进行排序运算，删除重复的记录再返回结果。实际大部分应用中是不会产生重复的记录，最常见的是过程表与历史表 UNION。
UNION ALL：The **UNION ALL** command is equal to the UNION command, except that UNION ALL selects all values.只是简单的将两个结果合并后就返回。这样，如果返回的两个结果集中有重复的数据，那么返回的结果集就会包含重复的数据了。
A **UNION** statement effectively does a SELECT DISTINCT on the results set. If you know that all the records returned are unique from your union, use UNION ALL instead, it gives faster results.从效率上说，UNION ALL 要比 UNION 快很多，所以，如果可以确认合并的两个结果集中不包含重复的数据的话，那么就使用 UNION ALL，

19. ******Several ways for Pagination

取出 sql 表中第 31 到 40 的记录（以自动增长 ID 为主键）
sql server 方案：select top 10 * from t where id not in (select top 30 id from t order by id) orde by id
oracle 方案：select * from (select rownum r,* from t where r<=40) where r>30

20. Inner Join vs. Left Outer Join vs. Right Outer Join vs. Full Outer Join



十、 Hibernate Interview Questions

1. ******What is ORM ?

ORM stands for **object/relational mapping.** ORM is the automated persistence of objects in a Java application to the tables in a relational database.

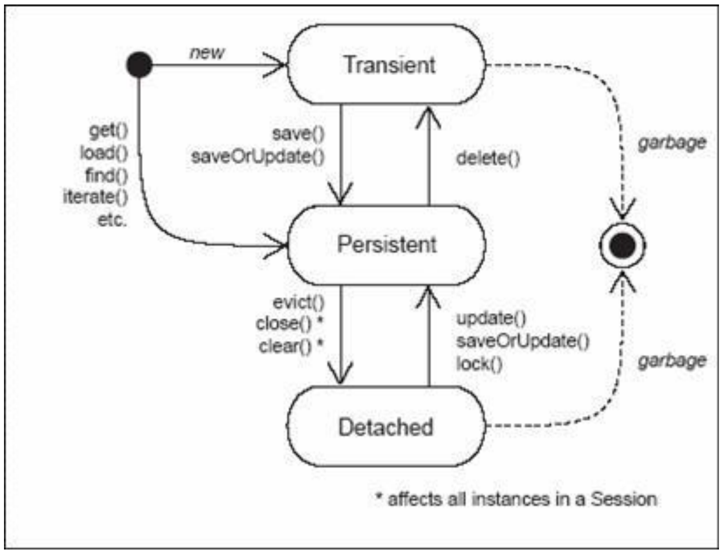
2. What is Hibernate?

Hibernate is a **pure Java object-relational mapping (ORM)** and persistence framework that allows you to **map plain old Java objects** to relational database tables using (XML) configuration files. Its purpose is to relieve the developer from a significant amount of relational data persistence-related programming tasks.

3. ******What are the Instance states in Hibernate? 就是 Hibernate session 的生命周期 life cycle

- **Transient**: The instance is not, and has never been associated with any persistence context. It has no persistent identity (primary key value). 使用 new 操作符初始化的对象不是立刻就持久的。它们的状态是瞬时的，也就是说它们没有任何跟数据库表相关联的行为，只要应用不再引用这些对象（不再被任何其它对象所引用），它们的状态将会丢失，并由垃圾回收机制回收。

- **Persistent**: The instance is currently associated with a persistence context. It has a persistent identity (primary key value) and, perhaps, a corresponding row in the database. For a particular persistence context, Hibernate guarantees that persistent identity is equivalent to Java identity (in-memory location of the object). 持久实例是任何具有数据库标识的实例。它有持久化管理器 Session 统一管理，持久实例是在事务中进行操作的——它们的状态在事务结束时同数据库进行同步。当事务提交时，通过执行 SQL 的 INSERT、UPDATE 和 DELETE 语句把内存中的状态同步到数据库中。
- **Detached:** The instance was once associated with a persistence context, but that context was closed, or the Instance was serialized to another process. It has a persistent identity and, perhaps, a corresponding row in the database. For detached instances, Hibernate makes no guarantees about the relationship between persistent dentity and Java identity. Session 关闭之后，持久化对象就变为离线对象。离线表示这个对象不能再与数据库保持同步，它们不再受 Hibernate 管理。



4. <u>********What is the advantage of Hibernate over jdbc? (这里可以说是 Hibernate 的优点)**</u>

Hibernate Vs. JDBC :-

1) **Hibernate** is data base independent, your code will work for all ORACLE,MySQL ,SQLServeretc. **JDBC** query must be data base specific.
2) As **Hibernate** is set of Objects , you don't need to learn SQL language. You can treat TABLE as a Object . Only Java knowledge is need. **JDBC** you need to learn SQL.
3) Don't need Query tuning in case of **Hibernate**. If you use Criteria Quires in Hibernate then **hibernate** automatically tuned your query and return best result with performance. In case of **JDBC** you need to tune your queries.
4) You will get benefit of Cache. **Hibernate** support two level of cache. First level and 2nd level. So you can store your data into Cache for better performance. In case of **JDBC** you need to implement your java cache .
5) **Hibernate** supports Query cache and It will provide the statistics about your query and database status. **JDBC** Not provides any statistics.
6) Development fast in case of **Hibernate** because you don't need to write queries
7) No need to create any connection pool in case of Hibernate. You can use c3p0. In case of **JDBC** you need to write your own connection pool

8) In the xml file you can see all the relations between tables in case of **Hibernate**. Easy readability.
9) You can load your objects on start up using lazy=false in case of **Hibernate**. **JDBC** Don't have such support.
10) **Hibernate** Supports automatic versioning of rows but **JDBC** Not

5. ******What is the general flow of Hibernate communication with RDBMS? 与远程数据库交互的工作流

The general flow of Hibernate communication with RDBMS is :

- **Load** the Hibernate configuration file and **create** configuration **object**. It will **automatically load** all hbm mapping files
- **Create** session factory from configuration object
- **Get one session** from this session factory
- **Create** HQL Query
- **Execute** query to get list containing Java objects

6. ******Difference between getCurrentSession() and openSession() in Hibernate ?

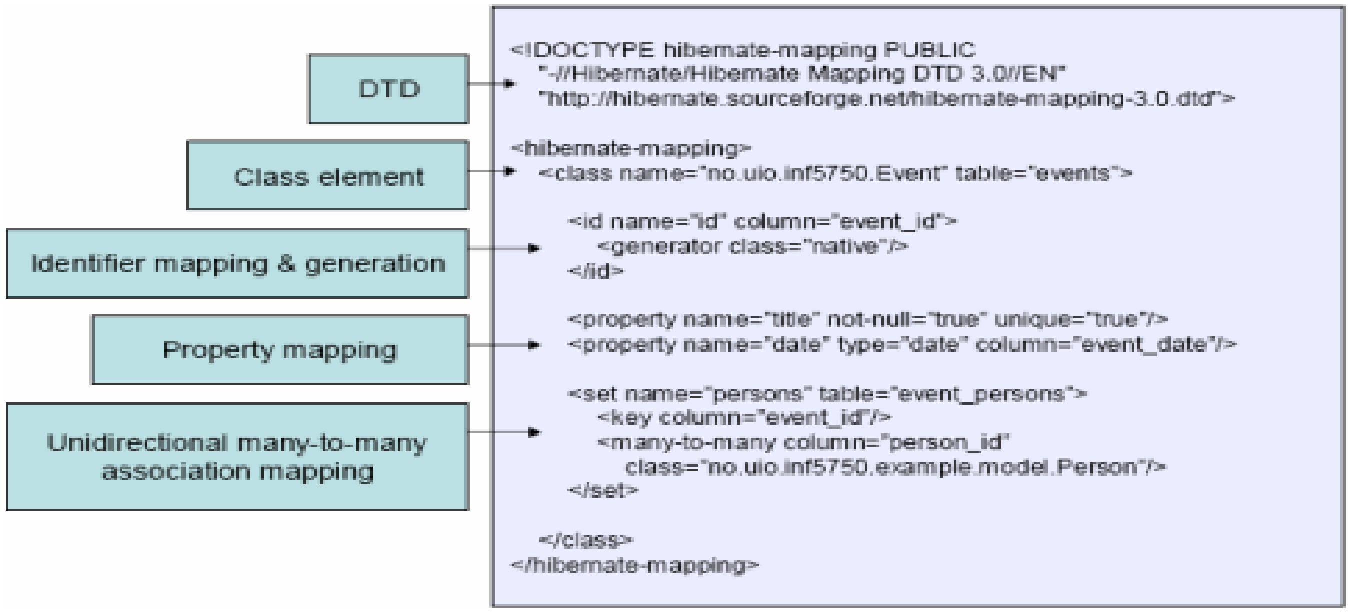
- **getCurrentSession() :** A Session is opened when **getCurrentSession**() is called for the **first time** and **closed** when the transaction ends. It is also flushed automatically before the transaction commits. You can call **getCurrentSession**() as often and anywhere you want as long as the transaction runs.如果有 session 就用这个存在的 session，如果没有就新建一个 session
- **openSession**() : If you decide to use manage the Session yourself the go for sf.**openSession**() , you have to **flush**() and **close**() it. It does **not** flush and close() automatically.

7. Why do you need ORM tools like hibernate?

The main advantage of ORM like hibernate is that it **shields developers from messy SQL.** Apart from this, ORM provides following **benefits**:
Improved productivity, High-level object-oriented API , Less Java code to write , No SQL to write , Improved performance, Sophisticated caching , Lazy loading , Eager loading , Improved maintainability, A lot less code to writej , Improved portability, ORM framework generates database-specific SQL for you ,

8. What is the need for Hibernate xml mapping file?

Hibernate mapping file tells Hibernate which tables and columns to use to load and store objects. Typical mapping file look as follows:


DTD
Class element
Identifier mapping & generation
Property mapping
Unidirectional many-to-many association mapping

```xml
<!DOCTYPE hibernate-mapping PUBLIC
   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="no.uio.inf5750.Event" table="events">

      <id name="id" column="event_id">
         <generator class="native"/>
      </id>

      <property name="title" not-null="true" unique="true"/>
      <property name="date" type="date" column="event_date"/>

      <set name="persons" table="event_persons">
         <key column="event_id"/>
         <many-to-many column="person_id"
            class="no.uio.inf5750.example.model.Person"/>
      </set>

   </class>
</hibernate-mapping>
```

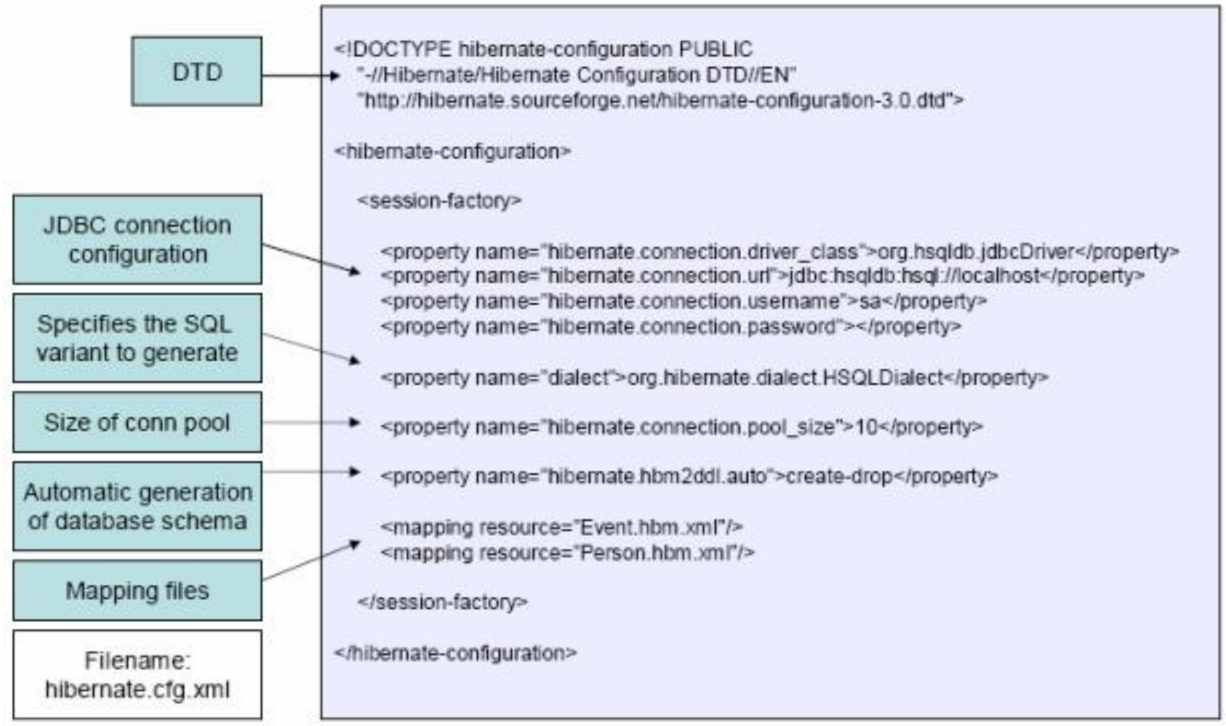
9. <u>What are the most common methods of Hibernate configuration?</u>

The most common methods of Hibernate configuration are:

- Programmatic configuration
- XML configuration (hibernate.cfg.xml)

10. <u>What are the important tags of hibernate.cfg.xml?</u>

Following are the important tags of hibernate.cfg.xml:


DTD
JDBC connection configuration
Specifies the SQL variant to generate
Size of conn pool
Automatic generation of database schema
Mapping files
Filename: hibernate.cfg.xml

```xml
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD//EN"
  "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

  <session-factory>

    <property name="hibernate.connection.driver_class">org.hsqldb.jdbcDriver</property>
    <property name="hibernate.connection.url">jdbc:hsqldb:hsql://localhost</property>
    <property name="hibernate.connection.username">sa</property>
    <property name="hibernate.connection.password"></property>

    <property name="dialect">org.hibernate.dialect.HSQLDialect</property>

    <property name="hibernate.connection.pool_size">10</property>

    <property name="hibernate.hbm2ddl.auto">create-drop</property>

    <mapping resource="Event.hbm.xml"/>
    <mapping resource="Person.hbm.xml"/>

  </session-factory>

</hibernate-configuration>
```

11. What are the Core interfaces are of Hibernate framework?

The **five core** interfaces are used in just about every Hibernate application. Using these interfaces, you can store and retrieve persistent objects and control transactions.
Session interface, SessionFactory interface , Configuration interface , Transaction interface , Query and Criteria interfaces

12. What role does the Session interface play in Hibernate?

The **Session interface** is the **primary interface** used by Hibernate applications. It is a single-threaded, short-lived object representing a conversation between the application and the persistent store. It allows you to create query objects to retrieve persistent objects.Session session = sessionFactory.openSession();
Session interface role:
Wraps a JDBC connection
Factory for Transaction
Holds a mandatory (first-level) cache of persistent objects, used when navigating the object graph or looking up objects by identifier

13. What role does the SessionFactory interface play in Hibernate?

The application obtains Session instances from a **SessionFactory**. There is typically a single SessionFactory for the whole application during application initialization. The **SessionFactory** caches generate SQL statements and other mapping metadata that Hibernate uses at runtime. It also holds cached data that has been read in one unit of work and may be reused in a future unit of work
SessionFactorysessionFactory = configuration.buildSessionFactory();

14. What is Hibernate Query Language (HQL)?

Hibernate offers **a query language** that embodies a very powerful and flexible mechanism to query, store, update, and retrieve objects from a database. This language, the **Hibernate query Language** (HQL), is an object-oriented extension to SQL.

15. How do you map Java Objects with Database tables?

First we need to write Java domain objects (beans with setter and getter).
Write **hbm.xml**, where we **map java class to table** and database columns to Java class variables.

16. ******What's the difference between load() and get()? 这个问题经常问

load() vs. get() :-

load()	get()
Only use the load() method if you are **sure** that the object exists.	If you are **not sure** that the object exists, then use one of the get() methods.
load() method will **throw an exception** if the unique id is not found in the database.	get() method will **return null** if the unique id is not found in the database.
load() just **returns a proxy** by default and database **won't be hit until** the proxy is first invoked.	get() will **hit the database immediately.**

17. What is the difference between merge and update ?

Use **update**() if you are sure that the session does not contain an already persistent instance **with** the same identifier, and **merge**() if you want to merge your modifications at any time **without** consideration of the state of the session.

18. Define cascade and inverse option in one-many mapping?

- **cascade** - enable operations to cascade to child entities.cascade="all|none|save-update|delete|all-delete-orphan"
- **inverse** - mark this collection as the "inverse" end of a bidirectional association.inverse="true|false" Essentially "inverse" indicates which end of a relationship should be ignored, so when persisting a parent who has a collection of children, should you ask the parent for its list of children, or ask the children who the parents are?

19. What are the benefits does HibernateTemplate provide?

The benefits of **HibernateTemplateare** :

- **HibernateTemplate**, a Spring Template class **simplifies** interactions with Hibernate Session.
- Common functions are **simplified** to single method calls.
- Sessions are **automatically closed**.
- Exceptions are **automatically caught** and converted to runtime exceptions.

20. How do you switch between relational databases without code changes?

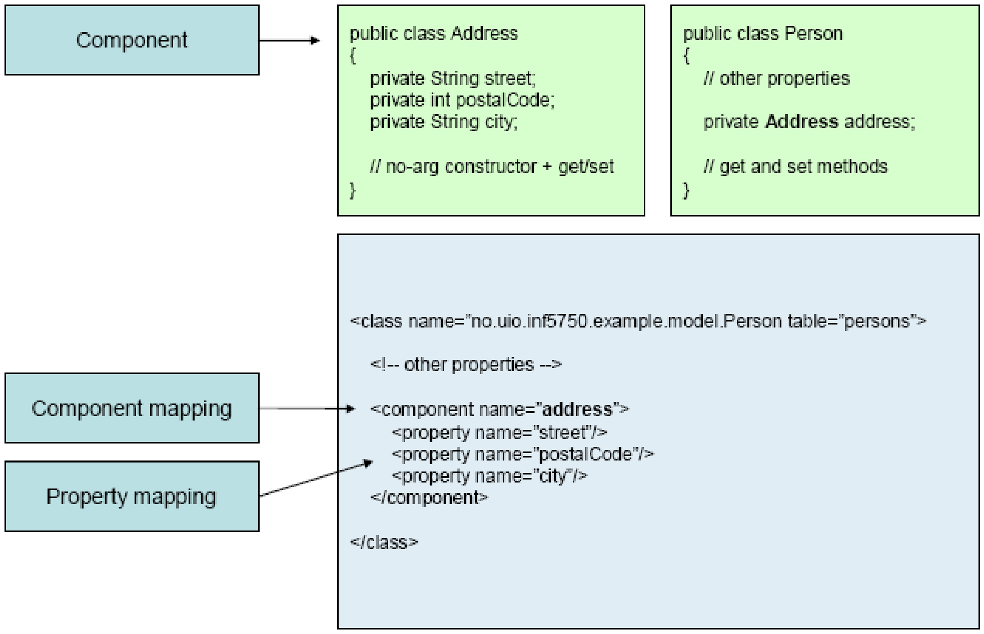
Using **Hibernate SQL Dialects** , we can switch databases. Hibernate will generate appropriate hql queries based on the dialect defined.

21. What are derived properties?

The properties that are **not** mapped to a column, **but** calculated at runtime by evaluation of an expression are called derived properties. The expression can be defined using the formula attribute of the element.

22. What is component mapping in Hibernate?

A component is an object saved **as a value**, not as a reference
A component can be **saved directly** without needing to declare interfaces or identifier properties
Required to define an empty constructor
Shared references not supported
Example:



```
public class Address
{
    private String street;
    private int postalCode;
    private String city;

    // no-arg constructor + get/set
}
```

```
public class Person
{
    // other properties

    private Address address;

    // get and set methods
}
```

```
<class name="no.uio.inf5750.example.model.Person table="persons">

    <!-- other properties -->

    <component name="address">
        <property name="street"/>
        <property name="postalCode"/>
        <property name="city"/>
    </component>

</class>
```

23. What is the difference between sorted and ordered collection in hibernate?

sorted collection vs. order collection :-

sorted collection	**order** collection
A sorted collection is sorting a collection by **utilizing** the sorting features provided by the Java collections framework. The sorting occurs in the memory of JVM which running Hibernate, **after** the data being read from database **using java comparator.**	Order collection is sorting a collection by **specifying** the order-by clause for sorting this collection when **retrieval**.
If your collection is **not large**, it will be **more efficient** way to sort it.	If your collection is **very large**, it will be **more efficient** way to order it .

24. What are the Collection types in Hibernate ?

Bag ; Set ; List ; Array ; Map

25. What is Hibernate proxy?

The **proxy** attribute enables **lazy initialization** of persistent instances of the class. Hibernate will initially return **CGLIB proxies** which implement the named interface. The actual persistent object will be loaded when a method of the proxy is invoked.

26. How can Hibernate be configured to access an instance variable directly and not through a setter method ?

By **mapping** the property with access="field" in Hibernate metadata(元数据). This forces hibernate to **bypass** the setter method and access the instance variable directly while initializing a newly loaded object.

27. How can a whole class be mapped as immutable（不变的）?

Mark the class as **mutable="false"** (Default is true),. This specifies that instances of the class are (not) mutable. Immutable classes, may **not be updated or deleted by the application**.

28. What is the use of dynamic-insert and dynamic-update attributes in a class mapping?

Criteria（标准） is a simplified API for retrieving entities by composing（组成） Criterion objects. This is a very convenient approach for functionality like "search" screens where there is a variable number of conditions to be placed upon the result set.
dynamic-update (defaults to false): Specifies that UPDATE SQL should be generated at runtime and **contain only** those columns **whose values have changed**
dynamic-insert (defaults to false): Specifies that INSERT SQL should be generated at runtime and **contain only** the columns whose **values are not null.**

29. What do you mean by fetching（抓取） strategy（策略）?

A **fetching strategy** is the strategy Hibernate will use for **retrieving associated objects** if the application needs to navigate the association. Fetch strategies may be **declared** in the **O/R mapping metadata**, or **over-ridden** by a particular **HQL** or **Criteria query**.

30. What is automatic dirty checking?

Automatic dirty checking is a feature that saves us the effort of explicitly asking Hibernate to **update the database** when we **modify** the state of an object inside a transaction.

31. What is transactional write-behind?

Hibernate uses a sophisticated algorithm to **determine** an **efficient ordering** that **avoids** database **foreign key constraint violations** but is still sufficiently predictable to the user. This feature is called transactional write-behind.

32. What are Callback interfaces?

Callback interfaces allow the application to **receive** a **notification** when something **interesting happens** to an object—for example, when an object is loaded, saved, or deleted. Hibernate applications don't need to implement these callbacks, but they're useful for implementing certain kinds of generic functionality.

33. What are the differences between EJB 3.0 & Hibernate

Hibernate Vs EJB 3.0 :-

Hibernate	EJB 3.0
Session–Cache or collection of loaded objects relating to a single unit of work	Persistence Context-Set of entities that can be managed by a given EntityManager is defined by a persistence unit
XDoclet Annotations used to support Attribute Oriented Programming	Java 5.0 Annotations used to support Attribute Oriented Programming
Defines HQL for expressing queries to the database	Defines EJB QL for expressing queries

Supports Entity Relationships through mapping files and annotations in JavaDoc	Support Entity Relationships through Java 5.0 annotations
Provides a Persistence Manager API exposed via the Session, Query, Criteria, and Transaction API	Provides and Entity Manager Interface for managing CRUD operations for an Entity
Provides callback support through lifecycle, interceptor, and validatable interfaces	Provides callback support through Entity Listener and Callback methods
Entity Relationships are unidirectional. Bidirectional relationships are implemented by two unidirectional relationships	Entity Relationships are bidirectional or unidirectional

34. ******Difference between session.save() ,   session.saveOrUpdate() and session.persist()?

- **session.save() :**   Save does an insert and will **fail** if the **primary key** is already **persistent**.
- **session.saveOrUpdate() :** saveOrUpdate does a **select first** to determine if it needs to do an insert or an update. Insert data if primary key not exist otherwise update data.
- **session.persist() :** Does the same like session.save().

But session.save() return Serializable object but **session.persis**t() return void.   **session.save**() returns the generated identifier (Serializable object)   and **session.persist**() doesn't.

35. ******What is lazy fetching in Hibernate? 就是延迟加载

Lazy fetching decides whether to load child objects while loading the Parent Object. You need to do this setting respective hibernate mapping file of the parent **class. Lazy = true (means not to load child)** By default the lazy loading of the child objects is true. This make sure that the child objects are **not loaded** unless they are explicitly invoked in the application by calling getChild() method on parent. 延迟加载就是当在真正需要数据的时候，才真正执行数据加载操作

36. How to prevent concurrent update   in Hibernate?

version checking used in hibernate when more then one thread trying to access same data. Hibernate automatically create/update the version number when you update/insert any row in the table

37. What are the general considerations or best practices for defining your Hibernate persistent classes?

1) You must have a **default no-argument constructor(非参数构造函数)** for your persistent classes and there should be **getXXX()** (i.e accessor/getter) and **setXXX**(i.e. mutator/setter) methods for all your persistable instance variables.
2) You should implement the **equals**() and **hashCode**() methods based on your business key and it is important **not** to **use** the **id** field in your equals() and hashCode() definition if the id field is a surrogate key (i.e. Hibernate managed identifier). This is because the Hibernate only generates and sets the field when saving the object.
3）It is recommended to **implement** the **Serializable** interface. This is potentially useful if you want to migrate around a **multi-processor cluster**.

4）The persistent class should **not** be **final** because if it is final then lazy loading cannot be used by creating proxy objects.

38. Difference between session.update() and session.lock() in Hibernate ?

- **Both** of these methods and **saveOrUpdate**() method **are** intended for reattaching a detached object.

The **session.lock**() method simply reattaches the object to the session without checking or updating the database on the assumption that the database in sync with the detached object. It is the best practice to use either **session.update**(..) or session.saveOrUpdate()（上面情况用 update）. Use **session.lock**() only if you are absolutely **sure** that the **detached object** is in sync with your detached object or if it does not matter because you will be overwriting all the columns that would have changed later on within the same transaction.

39. Difference between list() and iterate() in Hibernate?

If instances are already be in the session or second-level cache **iterate**() will give **better** performance. If they are not already cached, **iterate**() will be **slower** than **list**() and might **require** many **database hits** for a simple query.

40. What does session.refresh() do

It is possible to **re-load** an object and all its collections **at any time**, using the **refresh**() method. This is useful when database **triggers** are used to **initialize** some of the properties of the object

41. What is the main difference between Entity Beans and Hibernate （可以看做是 Hibernate 和 EJB 的区别）

1) In **Entity Bean** at a time we can interact with only one data Base. Where as in **Hibernate** we can able to establishes the connections to more than One Data Base. Only thing we need to write one more configuration file.
2) **EJB** need container like Weblogic, WebSphere but **hibernate** don't need. It can be run on tomcat.
3） **Entity Beans** does not support OOPS concepts where as **Hibernate** does.
4) **Hibernate** supports multi level caching, where as **Entity Beans** doesn't.
5) In **Hibernate** C3P0 can be used as a connection pool.
6) **Hibernate** is container independent. **EJB** not.

十一、 EJB Interview Questions & Answers

1. ******What are Enterprise Java Beans? 什么是 EJB

Enterprise Java Beans **(EJB)** is a specification which defines component architecture for developing **distributed systems**. Applications written using the Enterprise JavaBeans architecture are **reusable**, **scalable**, **transactional**, and **secure**. Enterprise Java Bean's allow the developer to only focus on implementing the business logic of the application.

2. ******How many types of Enterprise beans are there and what are they?

There are 2 types EJB's and they are:

1. Session Bean's

2. Message Driven Bean's(MDB's)

3. How many types of Entity beans are there and what are they?

There are 2 types Entity bean's and they are:

1. **Container Managed Persistence**(CMP) Entity Bean's

2. **Bean Managed Persistence**(BMP) Entity Bean's

4. ******How many types of Session beans are there and what are they? 还有 stateless sessionbean 和 stateful sessionbean 的区别

There are **2** types Session bean's and they are:
1. **Statefull** Session Bean's
2. **Stateless** Session Bean's

a) **Stateful beans** are also Persistent session beans. They are designed to service business processes that span multiple method requests or transactions.
Stateless beans are designed to service business process that last only for a single method call or request.

b) **Stateful session beans** remembers the previous requests and reponses.
Stateless session beans do not remember the previous request and responses.

c)**Stateful session** beans does not have pooling concept.
Stattless session bean instances are pooled.

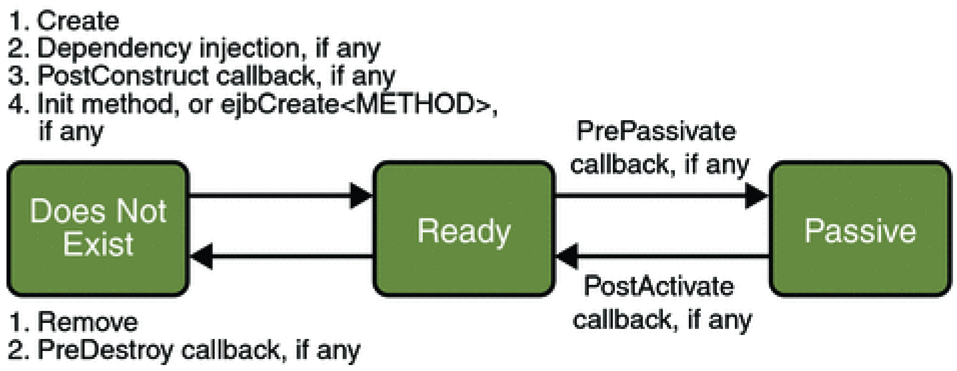
d) **Stateful S.Beans** can retain their state on behave of an individual client.
Stateless S.Beans donot maintain states.

e) **Stateful** S.Beans can be passivated and reuses them for many clients.
Stateless S.Beans, client specific data has to be pushed to the bean for each method invocation which result in increase of network traffic.

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。
******The life cycle of EJB 这个就是 ejb 所有的生命周期

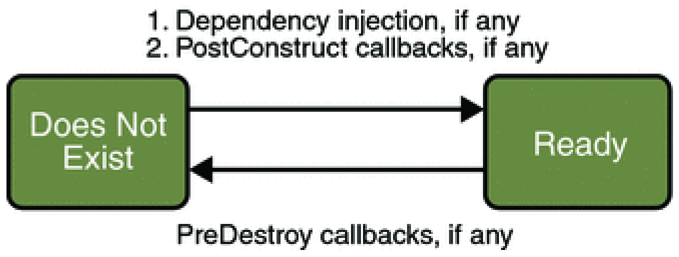
5. The Life Cycle of a Stateful Session Bean

a) The client **initiates** the life cycle by obtaining a reference to a stateful session bean. B)The container performs any dependency injection and then **invokes** the method annotated with @PostConstruct, if any. c) The bean is now **ready** to have its business methods invoked by the client. d) While in the ready stage, the EJB **container** may decide to deactivate, or **passivate**, the bean by **moving** it from memory to secondary storage. E)The EJB container invokes the method annotated @PrePassivate, if any, immediately before passivating it. f) If a client invokes a business method on the bean while it is in the passive stage, the EJB container activates the bean, calls the method annotated @PostActivate, if any, and then moves it to the ready stage. g)the client invokes a method annotated @Remove, and the EJB container calls the method annotated @PreDestroy, if any. The bean's instance is then ready for garbage collection.


1. Create
2. Dependency injection, if any
3. PostConstruct callback, if any
4. Init method, or ejbCreate<METHOD>, if any
PrePassivate callback, if any
Does Not Exist
Ready
Passive
PostActivate callback, if any
1. Remove
2. PreDestroy callback, if any

The Life Cycle of a Stateless Session Bean

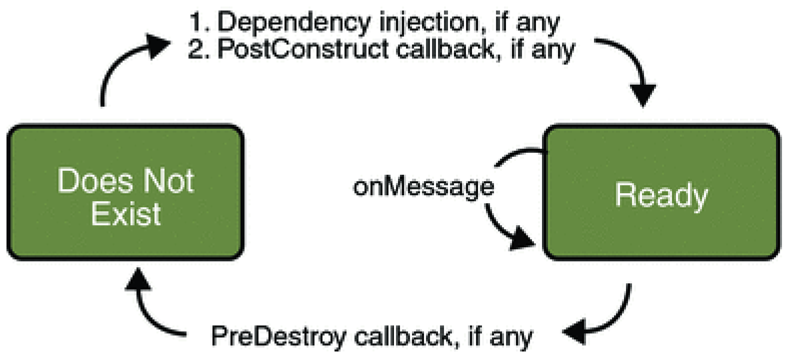
a)The client initiates the life cycle by **obtaining** a **reference** to a **stateless session** bean. B) The container performs any dependency injection and then **invokes** the method annotated @PostConstruct, if any. C)The bean is now **ready** to have its business methods invoked by the client. D)the EJB **container** calls the method annotated @PreDestroy, if any. The bean's instance is then ready for **garbage collection**.


1. Dependency injection, if any
2. PostConstruct callbacks, if any
Does Not Exist
Ready
PreDestroy callbacks, if any

The Life Cycle of a Message-Driven Bean

a)The EJB **container** usually **creates** a **pool** of message-driven bean instances. For each instance, the EJB container performs these tasks: b)If the message-driven bean uses dependency injection, the container injects these references before instantiating the instance. c)The container calls the method annotated @PostConstruct, if any.

Like a stateless session bean, a message-driven bean is never passivated, and it has only two states: nonexistent and ready to receive messages. D) the container calls the method annotated @PreDestroy, if any. The bean's instance is then ready for garbage collection.


1. Dependency injection, if any
2. PostConstruct callback, if any
Does Not Exist
onMessage
Ready
PreDestroy callback, if any

6. ******How is a enterprise bean different from a java bean?

Both the **enterprise bean** and the **java bean** are designed to be highly reusable. But other than being reusable there is no similarity between them. **Java bean** is mostly a simple client-side component whereas **enterprise bean** is a complex server side component.

What's difference between httpsession and EJB session bean ?

A session in a **Servlet**, is maintained by the **Servlet Container** through the **HttpSession** object, that is acquired through the request object. You cannot really instantiate a new HttpSession object, and it doesn't contains any business logic, **but** is more of a place where to store objects.

A session in **EJB** is maintained using the **SessionBeans**. You design beans that can contain business logic, and that can be used by the clients. You have two different session beans: **Stateful** and **Stateless**. The first one is somehow connected with a single client. It maintains the state for that client, can be used only by that client and when the client "dies" then the session bean is "lost".

7. How many java files should a developer code to develop a session bean?

3 java files has to be provided by the developer. They are:

1) an Home Interface

2) a Remote Interface

3) And a Session Bean implementation class.

8. Explain the role of Home Interface.

Home interface contains factory methods for locating, creating and removing instances of EJB's.

9. Explain the role of Remote Interface.

Remote interface defines the business methods callable by a client. All methods defined in the remote interface must throw RemoteException.

10. What is the need for a separate Home interface and Remote Interface. Can't they be defined in one interface?

EJB doesn't allow the client to directly communicate with an enterprise bean. The client has to use home and remote interfaces for any communication with the bean. The Home Interface is for communicating with the container for bean's life cycle operations like creating, locating,removing one or more beans. While the remote interface is used for remotely accessing the business methods.

11. What are callback methods?

Callback methods are bean's methods, which are called by the container. These are called to notify the bean, of it's life cycle events.

12. How will you make a session bean as stateful or stateless?

We have to specify the it in the deployment descriptor(ejb-jar.xml) using tag.

13. What is meant by Activation?

The process of transferring an enterprise bean from secondary storage to memory.

14. What is meant by Passivation?

The process of transferring an enterprise bean from memory to secondary storage.

15. What is a re-entrant Entity Bean?

An re-entrant Entity Bean is one that can handle multiple simultaneous, interleaved, or nested invocations which will not interfere with each other.

16. What is an EJBContext?

EJBContext is an object that allows an enterprise bean to **invoke services** provided by the container and to obtain the information about the caller of a client-invoked method.

17. Explain the role of EJB Container?

EJB Container implements the EJB component contract of the J2EE architecture. It provides a runtime environment for enterprise beans that includes security, concurrency, life cycle management, transactions, deployment, naming, and other services. An EJB Container is provided by an EJB Server.

18. What are Entity Bean's?

Entity Bean is an enterprise bean that represents persistent data maintained in a database. An entity bean can manage its own persistence or can delegate this function to its container. An entity bean is identified by a primary key. If the container in which an entity bean is hosted crashes, the entity bean, its primary key, and any remote references survive the crash.

19. What is a Primary Key?

Primary Key is an object that uniquely identifies an entity bean within a home.

20. What is a Deployment Descriptor?

Deployment Descriptor is a XML file provided with each module and application that describes how they should be deployed. The deployment descriptor directs a deployment tool to deploy a module or application with specific container options and describes specific configuration requirements that a deployer must resolve.

21. ******What are the difference's between a Local Interface and a Remote Interface?

Local Interfaces are new mechanism introduced in EJB 2.0 specification which enables components in the same container to bypass RMI and call each other's methods directly. In general, direct local method calls are faster than remote method calls. The downside is a loss of flexibility: because bean and client must run in the same container, the location of the bean is not transparent to the client (as it is with remote interfaces). remote interfaces pass parameters by value, while local interfaces pass them by reference.

22. What is a transaction?

A **transaction** is a sequence of operations that must all complete successfully, or leave system in the state it had been before the transaction started.

23. ******Explain about ACID properties of a transaction?

ACID is the acronym for the four properties guaranteed by transactions: atomicity, consistency, isolation, and durability.

Atomic : All or nothing. If a transaction is interrupted, all previous steps within that transaction are undone.

Consistent : The state of objects and/or the state of tables within a database move from one consistent state to another consistent state.

Isolated : What happens within one transaction should not affect or be visible within another transaction.
Durable : The effects of a transaction are persistent.

24. What are container managed transactions?

In an enterprise bean with **container-managed transactions**, the EJB container sets the boundaries of the transactions. You can use **container-managed transactions** with any type of enterprise bean: session, entity, or message-driven. **Container-managed transactions** simplify development because the enterprise bean code does not explicitly mark the transaction's boundaries. The code does not include statements that begin and end the transaction.

25. What are bean managed transactions?

For **bean-managed transactions**, the bean specifies transaction demarcations using methods in the javax.transaction.UserTransaction interface. Bean-managed transactions include any stateful or stateless session beans with a transaction-type set to Bean. Entity beans cannot use bean-managed transactions. For stateless session beans, the entering and exiting transaction contexts must match. For stateful session beans, the entering and exiting transaction contexts may or may not match. If they do not match, EJB container maintains associations between the bean and the nonterminated transaction. Session beans with bean-managed transactions cannot use the setRollbackOnly and getRollbackOnly methods of the javax.ejb.EJBContext interface.

26. What is the difference between container managed and bean managed transaction?

I**n container-managed transaction,** the transaction boundaries are defined by the container while in **bean managed transaction,** the transaction boundaries are defined by the bean. **Entity Beans** transactions are always **container managed**.

27. What is the difference between Container-Managed Persistent (CMP) bean and Bean-Managed Persistent(BMP) ?

Container-managed persistence(**CMP**) beans are the **simplest** for the bean developer to create and the **most difficult** for the EJB server to support. This is because all the logic for synchronizing the bean's state with the database is **handled automatically** by the container. This means that the bean developer **doesn't need** to write any data access logic, while the EJB server is supposed to take care of all the persistence needs automatically. With **CMP**, the container manages the persistence of the entity bean. A CMP bean developer **doesn't need** to worry about **JDBC** code and **transactions**, because the **Container** performs database calls and transaction management instead of the programmer. **Vendor tools** are used to **map** the entity fields to the database and absolutely **no database access** code is written in the bean class. All table mapping is specified in the deployment descriptor.
The bean-managed persistence (BMP) enterprise bean manages synchronizing its state with the database as **directed** by the **container**. The bean uses a database API to read and write its fields to the database, but the **container** tells it when to do each **synchronization** operation and manages the transactions for the bean automatically. **Bean-managed persistence** gives the bean developer the flexibility to perform persistence operations that are too complicated for the container or to use a data source that is not supported by the container. **BMP** beans are **not 100% database-independent**, because they may contain database-specific code, **but CMP** beans are **unable** to perform **complicated DML**

(data manipulation language) statements. EJB 2.0 specification introduced some new ways of querying database (by using the EJB QL - query language).

28. Why entity bean's transaction can't be managed by the bean?

Entity bean's represent the data and responsible for the integrity of the data. Entity bean's doesn't represent business operations to manage transactions. So there is no requirement for an entity bean managing it's transaction.

29. How many types of transaction attributes are available in EJB and what are they?

There 6 types of Transaction Atributes defined in EJB. They are as follows:
1. **NotSupported** : If the method is called within a transaction, this transaction is suspended during the time of the method execution.
2. **Required** : If the method is called within a transaction, the method is executed in the scope of this transaction; otherwise, a new transaction is started for the execution of the method and committed before the method result is sent to the caller.
3. **RequiresNew** : The method will always be executed within the scope of a new transaction. The new transaction is started for the execution of the method, and committed before the method result is sent to the caller. If the method is called within a transaction, this transaction is suspended before the new one is started and resumed when the new transaction has completed.
4. **Mandatory**: The method should always be called within the scope of a transaction, else the container will throw the TransactionRequired exception.
5. **Supports** : The method is invoked within the caller transaction scope; if the caller does not have an associated transaction, the method is invoked without a transaction scope.
6. **Never** : The client is required to call the bean without any transaction context; if it is not the case, a java.rmi.RemoteException is thrown by the container.

30. Can a client program directly access an Enterprise bean?

No. EJB Clients never access an EJB directly. The container insulates the beans from direct access from client applications. Every time a bean is requested, created, or deleted, the container manages the whole process.

31. When is an application said to be distributed?

An application is distributed when its components are running in separate runtime environments(JVM's), usually on different platforms connected via a network.Distributed applications are usually of 3 types.
They are :
1. two tier (client and a server)
2.three tier (client and a middleware and a server)
3. multitier (client and multiple middleware and multiple servers).

32. ******What is an EAR file?

EAR is Enterprise Archive file. A archive file that contains a J2EE application.

33. What do mean by business method?

A method of an enterprise bean that implements the business logic or rules of an application.

34. What is a finder method?

A method defined in the home interface and invoked by a client to locate an entity bean.

35. What is the difference between find and select methods in EJB?

A select method can return a persistent field (or a collection thereof) of a related entity bean. A finder method can return only a local or remote interface (or a collection of interfaces).Because it is not exposed in any of the local or remote interfaces, a select method cannot be invoked by a client. It can be invoked only by the methods implemented within the entity bean class. A select method is usually invoked by either a business or a home method.A select method is defined in the entity bean class. For bean-managed persistence, a finder method is defined in the entity bean class, but for container-managed persistence it is not.

36. Explain about setEntityContext method in Entity bean?

The setEntityContext() method is used to set the EntityContext interface for that bean. The EntityContext contains information about the context under which bean is operating. EntityContext interface gives security information about caller. The EntityContext is set only once in the life time of an entity bean instance.

37. What is the use of unsetEntityContext in Entity Bean?

The unsetEntityContext() method is called at the end of a beans life cycle before the instance is unloaded from memory. It is used to dereference EntityContext and to perform any clean up operations if required.

38. Explain ejbLoad and ejbStore methods of Entity Bean?

ejbLoad method is primarily used for data retrievals(检索). ejbStore is used for updating data. Typically the container invokes ejbLoad before the *first* business method in a transaction and the ejbStore is invoked at the *end* of the transaction. ejbStore method will be invoked when any values of the entity are changed as part of the transaction.

39. Difference between SessionBean remove() and EntityBean remove() method?

SessionBean remove() : inform the **container** of your loss of interest in this bean. **Container** will **remove** the instance.

EntityBean remove() : **delete** an **entity bean** without first instantiating it. **Delete** the row of the table using mentioned **primary key.**

40. What is the difference between a Coarse Grained Entity Bean and a Fine Grained Entity Bean?

A fine grained entity bean is pretty much directly mapped to one relational table, in third normal form. **A coarse grained entity bean** is larger and more complex, either because its attributes include values or lists from other tables, or because it owns one or more sets of dependent objects. **Note** that the **coarse grained** bean might be mapped to a **single table** or **flat file**, but that single **table** is going to be pretty ugly, with data copied from other tables, repeated field groups, columns that are dependent on **non-key fields**, etc.

Fine grained entities are generally considered a **liability** in large systems because they will tend to **increase** the load on several of the EJB servers subsystems (there will be more objects exported through the distribution layer, more objects participating in transactions, more skeletons in memory, more EJB Objects in memory,etc.)

41. What are the ways for a client application to get an EJB object?

1) The client has the **JNDI name** of the **EJB object**; this name is used to get the EJB object.
2) The client has the **JNDI name** of the **Home object**, this is a more usual case; this name is used to get the Home object, then a finder method is invoked on this Home to **obtain** one or several **entity bean objects**. The client may also invoke a "**create**" **method** on the Home object to **create** a **new EJB object** (session or entity).
3) The client has got a **handle** on an **EJB object**. A handle is an object that **identifies** an EJB **object**; it may be **serialized**, stored, and used later by a client to obtain a reference to the EJB Object, using the **getEJBObject** method().
4)The client has got a **reference** to an EJB **object**, and some methods defined on the remote interface of this Enterprise Bean return EJB objects.

42. What is handle and why it is used in EJB?

The **handle mechanism** allows a client application to **maintain** a **reference** to an **EJB object**. A **handle** object may be obtained by calling the **getHandle**() **method** on the reference to an EJB object. The main interest is that the **handle class** implements java.io.serializable interface, which means that a **handle** may be serialized. This allows the client to **store** the **handle**, or to pass it to another process. The **handle** may then be **deserialized** and used to obtain the reference to the EJB object, by calling the getEJBObject() method. **Handles** on session bean objects are valid **until** the session bean object exists. **Handles** on entity bean objects are valid during the complete life time of
the entity bean object; this means that such **handles** may be used by different clients and stored for a long time; the **EJB server** holding the entity bean **objects** may be stopped and restarted, the **handle** will still be valid.

十二、 WEB SERVICE Interview

1. ********WEB SERVICE 名词解释。JSWDL 开发包的介绍。JAXP、JAXM 的解释。SOAP、UDDI,WSDL 解释。**

Web Service:Web Service 是基于网络的、分布式的模块化组件，它执行特定的任务，遵守具体的技术规范，这些规范使得 Web Service 能与其他兼容的组件进行互操作。

- **JAXP**(Java API for XML Parsing) 定义了在 Java 中使用 DOM, SAX, XSLT 的通用的接口。这样在你的程序中你只要使用这些通用的接口，当你需要改变具体的实现时候也不需要修改代码。
- **JAXM**(Java API for XML Messaging) 是为 SOAP 通信提供访问方法和传输机制的 API。
- **WSDL** 是一种 XML 格式，用于将网络服务描述为一组端点，这些端点对包含面向文档信息或面向过程信息的消息进行操作。这种格式首先对操作和消息进行抽象描述，然后将其绑定到具体的网络协议和消息格式上以定义端点。相关的具体端点即组合成为抽象端点（服务）。
- **SOAP** 即简单对象访问协议(Simple Object Access Protocol)，它是用于交换 XML 编码信息的轻量级协议。
- **UDDI** 的目的是为电子商务建立标准；UDDI 是一套基于 Web 的、分布式的、为 Web Service 提供的、信息注册中心的实现标准规范，同时也包含一组使企业能将自身提供的 Web Service 注册，以使别的企业能够发现的访问协议的实现标准。
- A web service is a kind of software that is accessible on the Internet. It makes use of the XML messaging system and offers an easy to understand, interface for the end users.

2. ********Define Web Service?**

- Web services are application components
- Web services communicate using open protocols
- Web services are self-contained and self-describing
- Web services can be discovered using UDDI
- Web services can be used by other applications
- **XML** is the **basis** for Web services

3. ********Benefits of web services? Web services 的优点**

The **biggest advantage** of web service is that is supported by wide variety of platforms.

4. ********What is WSDL?**

- WSDL stands for **Web Services Description Language**
- WSDL is **written in XML** 说白了就是 **WSDL** 是个 **XML**
- WSDL is an **XML document**
- WSDL is used to **describe** Web services
- WSDL is also used to **locate** Web services

5. ********What is SOAP?**

- SOAP stands for **Simple Object Access Protocol**
- SOAP is a **communication protocol**
- SOAP is for **communication** between applications
- SOAP is a **format** for sending messages
- SOAP communicates via Internet
- SOAP is **platform independent**
- SOAP is **language independent**
- SOAP is **based** on **XML**
- SOAP is **simple** and **extensible**
- SOAP allows you to get around **firewalls**

6. **What is UDDI?**

- UDDI is a **directory** service where companies can register and search for Web services.
- UDDI stands for **Universal Description, Discovery and Integration**
- UDDI is a **directory** for **storing** information about web services
- UDDI is a **directory** of web service interfaces described by **WSDL**
- UDDI **communicates** via **SOAP**
- UDDI is built into the Microsoft .NET platform

7. ********What is REST?**

REST stands for **Representational State Transfer**. REST itself is not a standard, while it uses various standards such as HTTP, URL, XML/HTML/GIF/JPEG (Resource Representations) and text/xml, text/html, image/gif, image/jpeg, etc (MIME Types).

8. ********The difference between SOAP and REST**

- **SOAP** stands for Simple Object Access Protocol. **REST** stands for Representational State Transfer.

- **SOAP** is a XML based messaging protocol and **REST** is not a protocol but an architectural style.
- **SOAP** has a standard specification but there is none for **REST**.
- Even **SOAP** based web services can be implemented in **RESTful** style. **REST** is a concept that does **not** tie with any protocols.
- **SOAP** is distributed computing style and **REST** is web style (web is also a distributed computing model).
- **REST** messages should be **self-contained** and should help consumer in controlling the interaction between provider and consumer(example, links in message to decide the next course of action). But **SOAP** doesn't has any such requirements.
- **REST** does not enforces message format as XML or JSON or etc. But **SOAP** is XML based message protocol.
- **REST** follows stateless model. **SOAP** has specifications for stateful implementation as well.
- **SOAP** is strongly typed, has strict specification for every part of implementation. But **REST** gives the concept and less restrictive about the implementation.
- Therefore **REST** based implementation is simple compared to **SOAP** and consumer understanding.
- **SOAP** uses interfaces and named operations to expose business logic. **REST** uses (generally) URI and methods like (GET, PUT, POST, DELETE) to expose resources.
- **SOAP** has a set of standard specifications. WS-Security is the specification for security in the implementation. It is a detailed standard providing rules for security in application implementation. Like this we have separate specifications for messaging, transactions, etc. **REST** does not has dedicated concepts for each of these. **REST** predominantly relies on HTTPS. 就是 rest 无安全措施
- Above all both **SOAP** and **REST** depends on design and implementation of the application.

9. What kind of security is needed for web services?

The security level for web services should be more than that of what we say **Secure Socket Layer** (SSL). This level of security can be only achieved from **Entrust Secure Transaction** Platform. Web services need this level of security to ensure reliable transactions and secure confidential information .

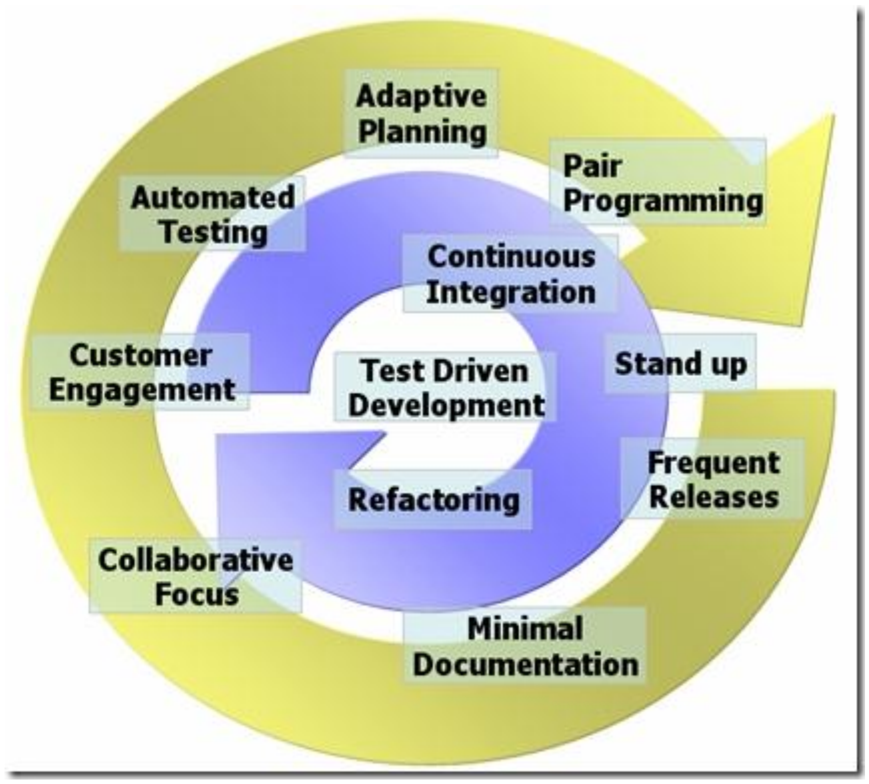
10. ******What tools are used to test a web service?

I have used **SoapUI** for SOAP WS and **Firefox poster plugin** for RESTFul Services.

十三、 软件工程的问题

1. Agile development（敏捷开发）

敏捷开发是一种以人为核心、迭代、循序渐进的开发方法。在敏捷开发中，软件项目的构建被切分成多个子项目，各个子项目的成果都经过测试，具备集成和可运行的特征。换言之，就是把一个大项目分为多个相互联系，但也可独立运行的小项目，并分别完成，在此过程中软件一直处于可使用状态。



Test-Driven Development，测试驱动开发。 (先写测试后写代码)

它是敏捷开发的**最重要的部分**。在 ThoughtWorks，我们实现任何一个功能都是从测试开 始，首先对业务需求进行分析，分解为一个一个的 Story，记录在 Story Card 上。然后两个人同时坐在电脑前面，一个人依照 Story，从业务需求的角度来编写测试代码，另一个人看着他并且进行思考，如果有不同的意见就会提 出来进行讨论，直到达成共识，这样写出来的测试代码就真实反映了业务功能需求。接着由另一个人控制键盘，编写该测试代码的实现。如果没有测试代码，就不能 编写功能的实现代码。先写测试代码，能够让开发人员明确目标，就是让测试通过。

Continuous Integration，持续集成。 （一上来就是在一起的开发）

在以往的软件开发过程中，集成是一件很痛苦的事情，通常很长时间才会做一次集成，这样的话，会引发很多问题，比如 build 未通过或者单元测试失败。敏捷开发中提倡持续集成，一天之内集成十几次甚至几十次，如此频繁的集成能尽量减少冲突，由于集成很频繁，每一次集成的改变也很少，即使集成失败也容易定位错误。一次集成要做哪些事情呢？它至少包括：获得所有源代码、编译源代码、运行所有测试，包括单元测试、功能测试 等；确认编译和测试是否通过，最后发送报告。当然也会做一些其它的任务，比如说代码分析、测试覆盖率分析等等。在我们公司里，开发人员的桌上有一个火山灯 用来标志集成的状态，如果是黄灯，表示正在集成；如果是绿灯，表示上一次集成通过，开发人员在这时候获得的代码是可用而可靠的；如果显示为红灯，就要小心 了，上一次集成未通过，需要尽快定位失败原因从而让灯变绿。在持续集成上，我们公司使用的是自己开发的产品 CruiseControl。

Refactoring，重构。 （每次检查代码也是重构的过程）

但是在敏捷开发中，重构贯穿于整个开发流程，每一次开发者 check in 代码之前，都要对所写代码进行重构，让代码达到 clean code that works。值得注意的是，在重构时，每一次改变要尽可能小，用单元测试来保证重构是否引起冲突，并且不只是对实现代码进行重构，如果测试代码中有重复， 也要对它进行重构。

Pair-Programming，结对编程。（两人一组）

在敏捷开发中，做任何事情都是 Pair 的，包括分析、写测试、写实现代码或者重构。Pair 做事有很多好处，两个人在一起探讨很容易产 生思想的火花，也不容易走上偏路。在我们公司，还有很多事都是 Pair 来做，比如 Pair 学习，Pair 翻译，Pair 做 PPT，关于这个话题，钱钱同学 有一篇很有名的文章对它进行介绍，名为 Pair Programming (结对编程)。

Stand up，站立会议。 （每天都要进行简短的会议讨论碰到的问题和进度）

每天早上，项目组的所有成员都会站立进行一次会议，由于是站立的，所以时间不会很长，一般来说是 15-20 分钟。会议的内容并不是需求 分析、任务分配等，而是每个人都回答三个问题：1. 你昨天做了什么？2. 你今天要做什么？ 3. 你遇到了哪些困难？站立会议让团队进行交流，彼此相互熟悉工作内容，如果有人曾经遇到过和你类似的问题，那么在站立会议后，他就会和你进行讨论。

Frequent Releases，小版本发布。 （频繁的发布版本，根据客户需求随时更改）

在敏捷开发中，不会出现这种情况，拿到需求以后就闭门造车，直到最后才将产品交付给客户，而是尽量多的产品发布，一般以周、月为单位。 这样，客户每隔一段时间就会拿到发布的产品进行试用，而我们可以从客户那得到更多的反馈来改进产品。正因为发布频繁，每一个版本新增的功能简单，不需要复 杂的设计，这样文档和设计就在很大程度上简化了。又因为简单设计，没有复杂的架构，所以客户有新的需求或者需求进行变动，也能很快的适应。

Minimal Documentation，较少的文档。 （文档跟着测试走，测试变了文档才变）

其实敏捷开发中并不是没有文档，而是有大量的文档，即测试。这些测试代码真实的反应了客户的需求以及系统 API 的用法，如果有新人加入团队，最快的熟悉项目的方法就是给他看测试代码，而比一边看着文档一边进行 debug 要高效。如果用书面文档或者注释，某天代码变 化了，需要对这些文档进行更新。一旦忘记更新文档，就会出现代码和文档不匹配的情况，这更加会让人迷惑。而在敏捷中并不会出现，因为只有测试变化了，代码 才会变化，测试是真实反应代码的。这时有人会问：代码不写注释行吗？一般来说好的代码不是需要大量的注释吗？其实简单可读的代码才是好的代码，既然简单可 读了，别人一看就能够看懂，这时候根本不需要对代码进行任何注释。若你觉得这段代码不加注释的话别人可能看不懂，就表示设计还不够简单，需要对它进行重 构。

Collaborative Focus，以合作为中心，表现为代码共享。 （互相熟悉对方的代码）

在敏捷开发中，代码是归团队所有而不是哪些模块的代码属于哪些人，每个人都有权利获得系统任何一部分的代码然后修改它，如果有人看到某 些代码不爽的话，那他能够对这部分代码重构而不需要征求代码作者的同意，很可能也不知道是谁写的这部分代码。这样每个人都能熟悉系统的代码，即使团队的人 员变动，也没有风险。

Customer Engagement ，现场客户。 （与客户一起工作）

敏捷开发中，客户是与开发团队一起工作的，团队到客户现场进行开发或者邀请客户到团队公司里来开发。如果开发过程中有什么问题或者产品经过一个迭代后，能够以最快速度得到客户的反馈。

Automated Testing ，自动化测试。

为了减小人力或者重复劳动，所有的测试包括单元测试、功能测试或集成测试等都是自动化的，这对 QA 人员提出了更高的要求。他们要熟悉开 发语言、自动化测试工具，能够编写自动化测试脚本或者用工具录制。我们公司在自动化测试上做了大量的工作，包括 Selenium 开源项目。

Adaptive Planning，可调整计划。 （随时修改）

敏捷开发中计划是可调整的，并不是像以往的开发过程中，需求分析->概要设计->详细设计->开发 ->测试->交付，每一个阶段都是有计划的进行，一个阶段结束便开始下一个阶段。而敏捷开发中只有一次一次的迭代，小版本的发布，根据客户反 馈随时作出相应的调整和变化。

敏捷开发过程与传统的开发过程有很大不同，在这过程中，团队是有激情有活力的，能够适应更大的变化，做出更高质量的软件。

2. Difference between Agile and waterfall and iterator

- **对比 iterator 方法**

相比迭代式开发两者都强调在较短的开发周期提交软件，**Agile** 方法的周期可能更短，并且更加强调队伍中的高度协作。

- **对比 waterfall**

两者没有很多的共同点，waterfall 式是最典型的预见性的方法，严格遵循预先计划的需求、分析、设计、编码、测试的步骤顺序进行。步骤成果作为衡量进度的方法，例如需求规格，设计文档，测试计划和代码审阅等等。

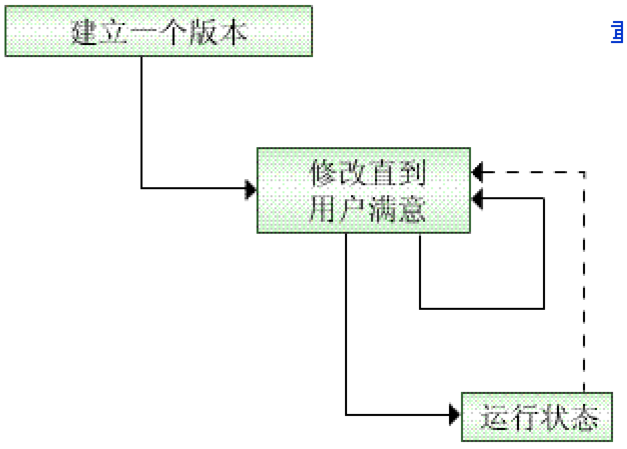
waterfall 的主要的问题是它的严格分级导致的自由度降低，项目早期即作出承诺导致对后期需求的变化难以调整，代价高昂。瀑布式方法在需求不明并且在项目进行过程中可能变化的情况下基本是不可行的。

相对来讲，**Agile** 方法则在几周或者几个月的时间内完成相对较小的功能，强调的是能将尽早将尽量小的可用的功能交付使用，并在整个项目周期中持续改善和增强。

有人可能在这样小规模的范围内的每次迭代中使用瀑布式方法，另外的人可能将选择各种工作并行进行，例如极限编程。

3. Software Development Model

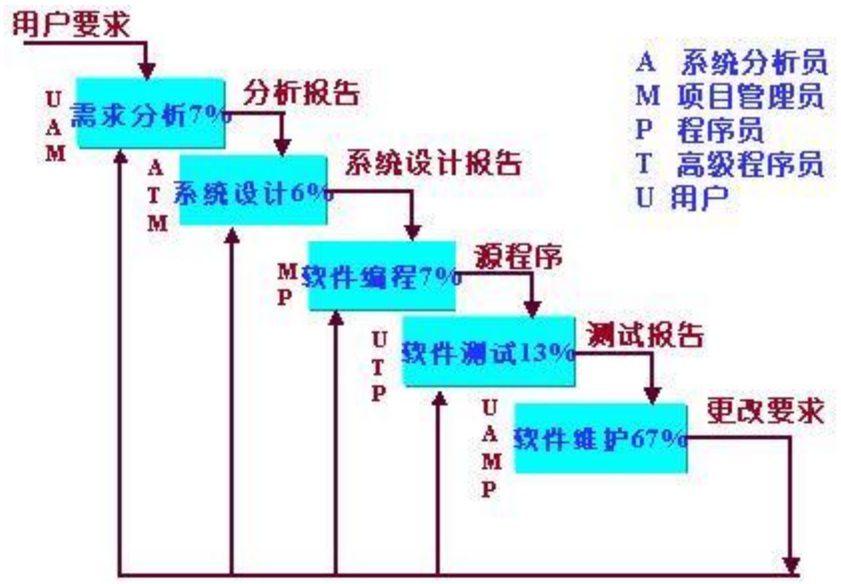
1. 边做边改模型（Build-and-Fix Model）；



这是一种类似作坊的开发方式，对编写几百行的小程序来说还不错，但这种方法对任何规模的开发来说都是不能令人满意的，其主要问题在于：

（1） 缺少规划和设计环节，软件的结构随着不断的修改越来越糟，导致无法继续修改；
（2） 忽略需求环节，给软件开发带来很大的风险；
（3） 没有考虑测试和程序的可维护性，也没有任何文档，软件的维护十分困难。

2. 瀑布模型（Waterfall Model）；



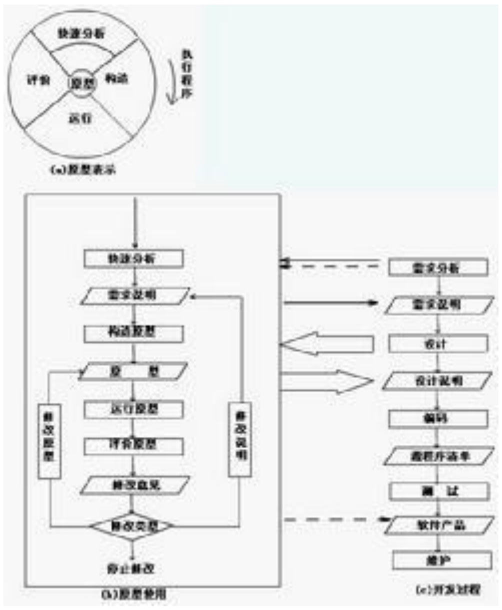
瀑布模型强调文档的作用，并要求每个阶段都要仔细验证。但是，这种模型的线性过程太理想化，已不再适合现代的软件开发模式，几乎被业界抛弃，其主要问题在于：

（1） 各个阶段的划分完全固定，阶段之间产生大量的文档，极大地增加了工作量；

（2） 由于开发模型是线性的，用户只有等到整个过程的末期才能见到开发成果，从而增加了开发的风险；

（3） 早期的错误可能要等到开发后期的测试阶段才能发现，进而带来严重的后果。

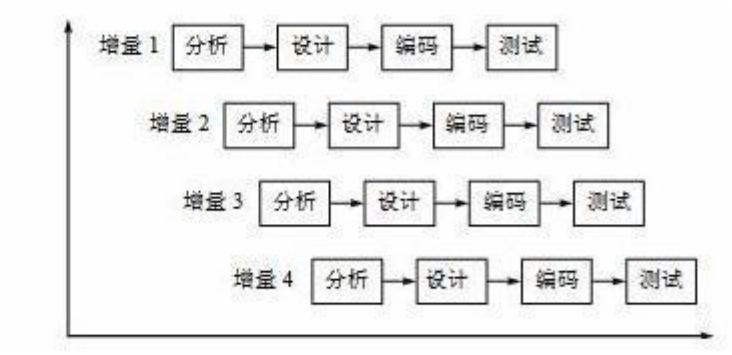
3. 快速原型模型（Rapid Prototype Model）；



快速原型模型的第一步是建造一个快速原型，实现客户或未来的用户与系统的交互，用户或客户对原型进行评价，进一步细化待开发软件的需求。 快速原型通过逐步调整原型使其满足客户的要求，开发人员可以确定客户的真正需求是什么；第二步则在第一步的基础上开发客户满意的软件产品。显然，快速原型 方法可以克服瀑布模型的缺点，减少由于软件需求不明确带来的开发

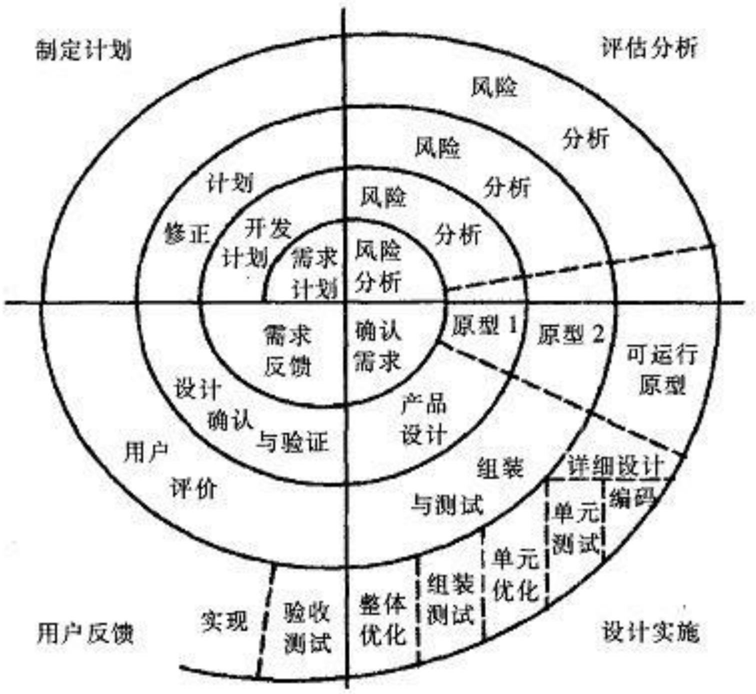
风险，具有显著的效果。快速原型的关键在于尽可能快速地建造出软件原型，一旦确定了客户的真正需求，所建造的原型将被丢弃。因此，原型系统的内部结构并不重要，重要的是必须迅速建立原型，随之迅速修改原型，以反映客户的需求。

4. 增量模型（Incremental Model）（演化模型(incremental model)）;



整个产品被分解成若干个构件，开发人员逐个构件地交付产品，这样做的好处是软件开发可以较好地适应变化，客户可以不断地看到所开发的软件，从而降低 开发风险。但是，增量模型也存在以下缺陷：（1） 由于各个构件是逐渐并入已有的软件体系结构中的，所以加入构件必须不破坏已构造好的系统部分，这需要软件具备开放式的体系结构。（2） 在开发过程中，需求的变化是不可避免的。增量模型的灵活性可以使其适应这种变化的能力大大优于瀑布模型和快速原型模型，但也很容易退化为边做边改模型，从 而是软件过程的控制失去整体性。

5.螺旋模型（Spiral Model）;



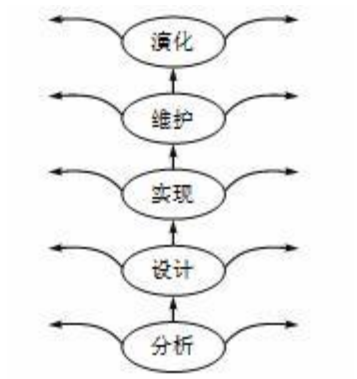
它将瀑布模型和快速原型模型结合起来，强调了其他模型所忽视的风险分析，特别适合于大型复杂的系统。

螺旋模型沿着螺线进行若干次迭代，图中的四个象限代表了以下活动：

（1） 制定计划：确定软件目标，选定实施方案，弄清项目开发的限制条件；

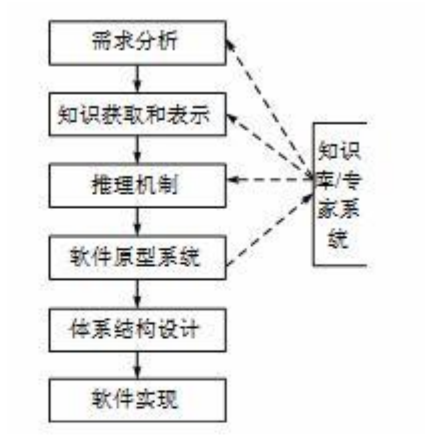
（2） 风险分析：分析评估所选方案，考虑如何识别和消除风险；
（3） 实施工程：实施软件开发和验证；
（4） 客户评估：评价开发工作，提出修正建议，制定下一步计划。
螺旋模型由风险驱动，强调可选方案和约束条件从而支持软件的重用，有助于将软件质量作为特殊目标融入产品开发之中。但是，螺旋模型也有一定的限制条件，具体如下：
（1） 螺旋模型强调风险分析，但要求许多客户接受和相信这种分析，并做出相关反应是不容易的，因此，这种模型往往适应于内部的大规模软件开发。
（2） 如果执行风险分析将大大影响项目的利润，那么进行风险分析毫无意义，因此，螺旋模型只适合于大规模软件项目。
（3） 软件开发人员应该擅长寻找可能的风险，准确地分析风险，否则将会带来更大的风险

6.喷泉模型(fountain model)；



(也称面向对象的生存期模型, OO 模型) 喷泉模型与传统的结构化生存期比较，具有更多的增量和迭代性质，生存期的各个阶段可以相互重叠和多次反复，而且在项目的整个生存期中还可以嵌入子生存期。

7.智能模型(四代技术（4GL）)；



智能模型拥有一组工具（如数据查询、报表生成、数据处理、屏幕定义、代码生成、高层图形功能及电子表格等），每个工具都能使开发人员在高层次上定义软件的某些特性，并把开发人员定义的这些软件自动地生成为源代码。
这种方法需要四代语言（4GL）的支持。4GL 不同于三代语言，其主要特征是用户界面极端友好，即使没有受过训练的非专业程序员，也能用它编写程 序；它是一种声明式、交互式和非过程性编程语言。4GL 还具有高效的程序代码、智能缺省假设、完备的数据库和应用程序生成器。目前

市场上流行的 4GL（如 Foxpro 等）都不同程度地具有上述特征。但 4GL 目前主要限于事务信息系统的中、小型应用程序的开发。

8.混合模型（hybrid model）

过程开发模型又叫混合模型（hybrid model），或元模型（meta-model）,把几种不同模型组合成一种混合模型，它允许一个项目能沿着最有效的路径发展，这就是过程开发模型（或混合模型）。

各种模型的优点和缺点

瀑布模型 文档驱动 系统可能不满足客户的需求 快速原型模型 关注满足客户需求 可能导致系统设计差、效率低，难于维护 增量模型 开发早期反馈及时，易于维护 需要开放式体系结构，可能会设计差、效率低 螺旋模型 风险驱动 风险分析人员需要有经验且经过充分训练

4. Design Patterns

这个另看保存的文件

5. RAD

Rapid Application development。

In rapid application development, structured techniques and prototyping are especially used to define users' **requirements** and to design the final system. The development process starts with the **development of preliminary data models** and **business process models** using structured techniques. In the next stage, requirements are verified using prototyping, eventually to refine the data and process models. These stages are repeated iteratively; further development results in "a combined business requirements and technical design statement to be used for constructing new systems".

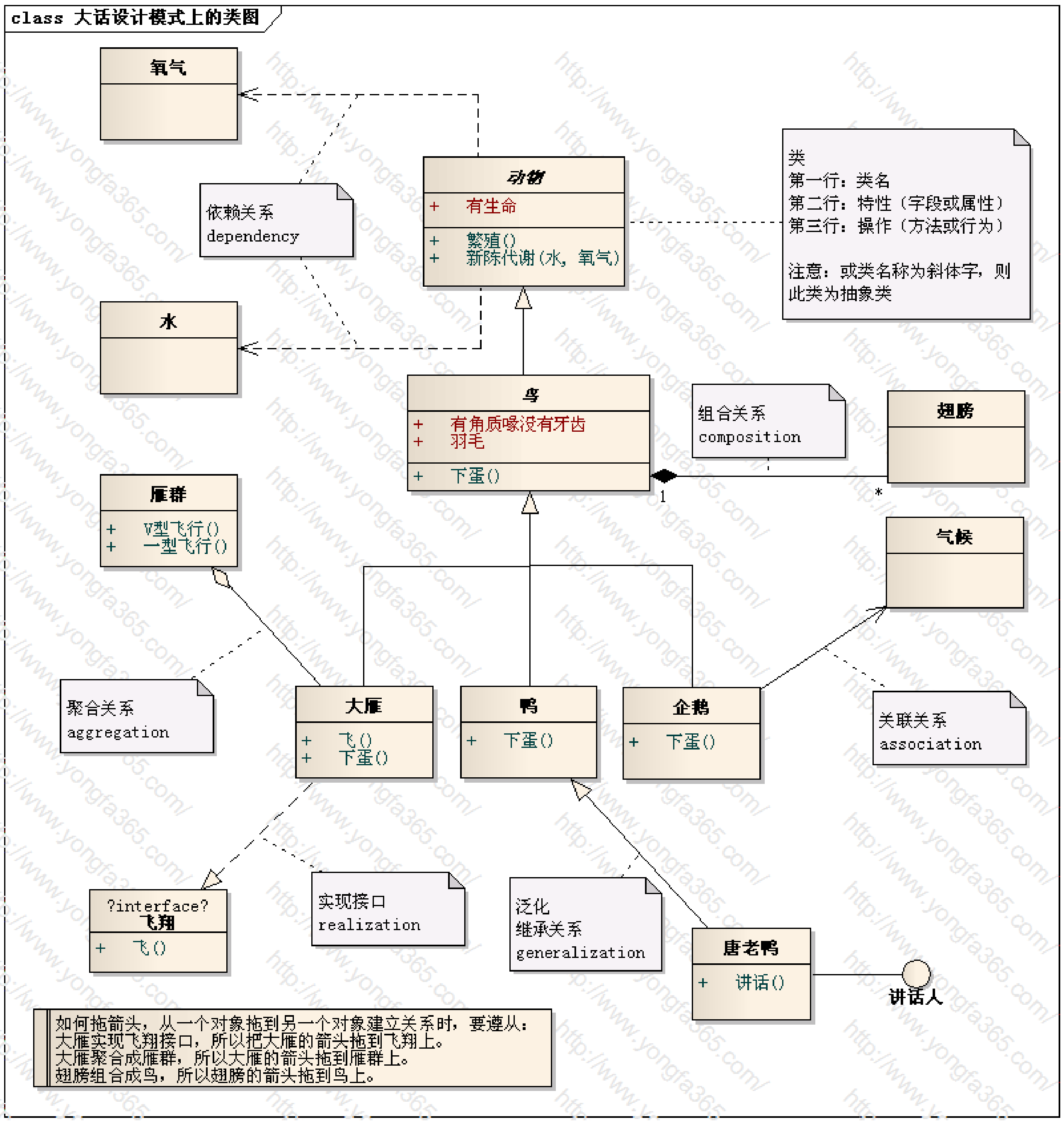
Requirements Planning phase – **combines** elements of the system planning and systems analysis phases of the System Development Life Cycle (SDLC). Users, managers, and IT staff members discuss and agree on business needs, project scope, constraints, and system requirements. It ends when the team agrees on the key issues and obtains management authorization to continue.（需求分析阶段，所有人员一起敲定大致关键点）

User design phase – during this phase, users interact with systems analysts and develop models and prototypes that represent all system processes, inputs, and outputs. The RAD groups or subgroups typically use a combination of Joint Application Development (JAD) techniques and CASE tools to translate user needs into working models. *User Design* is a continuous interactive process that allows users to understand, modify, and eventually approve a working model of the system that meets their needs.（协助 client 简历合适的系统模型）

Construction phase – focuses on program and application development task similar to the SDLC. In RAD, however, users continue to participate and can still suggest changes or improvements as actual screens or reports are developed. Its tasks are programming and application development, coding, unit-integration and system testing.（主要是 coding，但是 user 任然能修改需求，并根据需求进行 code 直到整合测试）

Cutover phase – resembles the final tasks in the SDLC implementation phase, including data conversion, testing, changeover to the new system, and user training. Compared with traditional methods, the entire process is compressed. As a result, the new system is built, delivered, and placed in operation much sooner. Its tasks are data conversion, full-scale testing, system changeover, user training.（最后的转换，user trainning，final testing）

6. Dependency, association, aggregation, composition, realization, generalization,



依赖关系	dependency

关联关系	association
聚合关系	aggregation
组合关系	composition
实现	realization
泛化 （就是 inheritance	generalization

association ： 连接模型元素及链接实例，用一条实线来表示；
dependency：表示一个元素以某种方式依赖于另一个元素，用一条虚线加箭头来表示；
aggregation：表示整体与部分的关系，用一条实线加空心菱形来表示；
composition：表示整体与部分的有一关系，用一条实线加实心菱形来表示；
generalization（inheritance）：表示一般与特殊的关系，用一条实线加空心箭头来表示；
realization：表示类与接口的关系，用一条虚线加空心箭头来表示；
注意：generalization 关系和 realization 关系又统称为一般关系；
总之：一般关系表现为 inheritance 或 realization(is a)，association 关系、aggregation 关系、合成/composition 表现为成员变量(has a)，dependency 关系表现为函数中的参数(use a)；
UML 中的 6 大关系详细说明：

1、association 关系：

含义：类与类之间的连结，association 关系**使一个类知道另外一个类的属性和方法**；通常含有“知道”，“了解”的含义
体现：在 C#中，association 关系是通过**成员变量**来实现的；
方向：双向或单向；
图示：实线 ＋ 箭头；箭头指向被关联的类；
举例：“渔民”需要知道“天气”情况才能够出海

```
        //公司关联雇员
public class Company
{
    private Employee employee;
    public Employee GetEmployee()
    {
        return employee;
    }
    public void SetEmployee(Employee employee)
    {
        this.employee = employee;
    }
    //公司运作
    public void Run()
    {
        employee.StartWorking();
    }
}

//A 关联 B
class A
{
    B b = new B();
}
class B
{
}
```

2、dependency 关系：

含义：是类与类之间的连接，表示一**个类依赖于另外一个类的定义**；dependency 关系仅仅描述了类与类之间的一种使用与被使用的关系；

体现：在 C#中体现为局部变量、方法/函数的参数或者是对静态方法的调用；

方向：单向；

图示：虚线 + 箭头；

举例：人依赖于水和空气；汽车依赖汽油

```
            //人划船，人依赖于船
public class Person
{
    //划船
    public void Oarage(Boat boat)
    {
        boat.Oarage();
    }
}

//A 依赖于 B
class A
{
    public void Function(B b)
    { }
}
class B
{
}
```

3、aggregation 关系：

含义：是 association 关系的一种，是一种强关联关系；aggregation 关系是整体和个体/部分之间的关系；association 关系的两个类处于同一个层次上，而 aggregation 关系的两个类处于不同的层次上，一个是整体，一个是个体/部分；在 aggregation 关系中，代表个体/部分的对象有可能会被多个代表整体的对象所共享；

体现：C++中，aggregation 关系通过将被聚合者的（数组）指针作为内部成员来实现的；
方向：单向；
图示：空心菱形 ＋ 实线 ＋ 箭头；箭头指向被聚合的类，也就是说，箭头指向个体/部分；
举例：鸭群与鸭子具有 aggregation 关系；汽车由引擎、轮胎以及其它零件组成，因为汽车坏掉了，没有坏掉的引擎，轮胎和其他零件还可以继续使用。

4、composition：

含义：它也是 association 关系的一种，但它是比 aggregation 关系更强的关系.composition 要求 aggregation 关系中代表整体的对象要负责代表个体/ 部分的对象的整个生命周期；composition 不能共享；在 composition 中，如果代表整体的对象被销毁或破坏，那么代表个体/部分的对象也一定会被销毁或破坏，而聚在合 关系中，代表个体/部分的对象则有可能被多个代表整体的对象所共享，而不一定会随着某个代表整体的对象被销毁或破坏而被销毁或破坏；
体现：在 C#中，composition 是通过成员变量来实现的；
方向：单向；
图示：实心菱形 ＋ 实线 ＋ 箭头；箭头指向代表个体/部分的对象，也就是被组合的类的对象；
举例：一个人由头、四肢、等各种器官组成，因为人与这些器官具有相同的生命周期，人死了，这些器官也挂了；

5、generalization 关系：

含义：它表示一个更 generalization 的元素和一个更具体的元素之间的关系；也就是通常所说的类的 inheritance 关系；
体现：在 C#中，generalization 关系通过类的 inheritance 来实现的；
方向：单向；子类 inheritance 父类；
图示：空心箭头 ＋ 实线；箭头指向父类；
举例：动物下面可以分为哺乳动物，脊椎动物，爬行动物等

6、realization 关系：

含义：它指定了两个实体之间的一份合同；即：一个实体定义一份合同，另外一个实体则保证履行该合同；
体现：在 C#中，realization 关系通过类实现接口来实现的，即：一个类实现某个接口；
方向：单向；子类实现接口；
图示：空心箭头 ＋ 虚线；箭头指接口向接口；
举例：唐老鸭（对象）会说话（接口），因为一般鸭子不会说话，所以不会将说话这个方法给一般的鸭子带上；超人（对象）会飞（接口）

十四、 Junit Interview

1. JUnit 是什么

JUnit 是一个开发源代码的 Java **测试框架**，用于编写和运行可重复的测试。他是用于**单元测试**框架体系 xUnit 的一个实例（用于 java 语言）。它包括以下特性：
1、用于测试期望结果的断言（Assertion）
2、用于共享共同测试数据的测试工具
3、用于方便的组织和运行测试的测试套件

4、图形和文本的测试运行器

要说明的是 junit 一般是用来进行单元测试的，因此需要了解被测试代码的内部结构（即所谓的白盒测试），另外 junit 是在 xp 编程和重构 （refactor）中被极力推荐使用的工具，因为在实现自动单元测试的情况下可以大大的提高开发的效率，但是实际上编写测试代码也是需要耗费很多的时间 和精力的，那么使用这个东东好处到底在哪里呢？笔者认为是这样的：

1、对于 xp 编程而言，要求在编写代码之前先写测试，这样可以强制 你在写代码之前好好的思考代码（方法）的功能和逻辑，否则编写的代码很不稳定，那么你需要同时维护测试代码和实际代码，这个工作量就会大大增加。因此在 xp 编程中，基本过程是这样的：构思－》编写测试代码－》编写代码－》测试，而且编写测试和编写代码都是增量式的，写一点测一点，在编写以后的代码中如果 发现问题可以较块的追踪到问题的原因，减小回归错误的纠错难度

2、对于重构而言，其好处和 xp 编程中是类似的，因为重构也是要求改一点测一点，减少回归错误造成的时间消耗。

3、 对于非以上两种情况，我们在开发的时候使用 junit 写一些适当的测试也是有必要的，因为一般我们也是需要编写测试的代码的，可能原来不是使用的 junit，如果使用 junit，而且针对接口（方法）编写测试代码会减少以后的维护工作，例如以后对方法内部的修改（这个就是相当于重构的工作了）。另 外就是因为 junit 有断言功能，如果测试结果不通过会告诉我们那个测试不通过，为什么，而如果是想以前的一般做法是写一些测试代码看其输出结果，然后再 由自己来判断结果使用正确，使用 junit 的好处就是这个结果是否正确的判断是它来完成的，我们只需要看看它告诉我们结果是否正确就可以了，在一般情况下 会大大提高效率。

2. JUnit 入门

cherami 整理

安装 JUnit

安装很简单，先到以下地址下载一个最新的 zip 包：

http://download.sourceforge.net/junit/

下载完以后解压缩到你喜欢的目录下，假设是 JUNIT_HOME，然后将 JUNIT_HOME 下的 junit.jar 包加到你的系统的 CLASSPATH 环境变量中，对于 IDE 环境，对于需要用到的 junit 的项目增加到 lib 中，其设置不同的 IDE 有不同的设置，这里不多讲。

如何使用 JUnit 写测试？

最简单的范例如下：

1、创建一个 TestCase 的子类:

```
package junitfaq;
import java.util.*;
import junit.framework.*;
public class SimpleTest extends TestCase {
public SimpleTest(String name) {
super(name);
}
```

2、写一个测试方法断言期望的结果：

```
public void testEmptyCollection() {
Collection collection = new ArrayList();
assertTrue(collection.isEmpty());
}
```

注意：JUnit 推荐的做法是以 test 作为待测试的方法的开头，这样这些方法可以被自动找到并被测试。

3、写一个 suite()方法，它会使用反射动态的创建一个包含所有的 testXxxx 方法的测试套件：

```
public static Test suite() {
return new TestSuite(SimpleTest.class);
}
```

4、写一个 main()方法以文本运行器的方式方便的运行测试：

```
public static void main(String args[]) {
junit.textui.TestRunner.run(suite());
}
}
```

5、运行测试：

以文本方式运行：

```
java junitfaq.SimpleTest
```

通过的测试结果是：

```
.
Time: 0
OK (1 tests)
```

Time 上的小点表示测试个数，如果测试通过则显示 OK。否则在小点的后边标上 F，表示该测试失败。

每次的测试结果都应该是 OK 的，这样才能说明测试是成功的，如果不成功就要马上根据提示信息进行修正了。

如果 JUnit 报告了测试没有成功，它会区分失败（failures）和错误（errors）。失败是你的代码中的 assert 方法失败引起的；而错误则是代码异常引起的，例如 ArrayIndexOutOfBoundsException。

以图形方式运行：

```
java junit.swingui.TestRunner junitfaq.SimpleTest
```

通过的测试结果在图形界面的绿色条部分。

以上是最简单的测试样例，在实际的测试中我们测试某个类的功能是常常需要执行一些共同的操作，完成以后需要销毁所占用的资源（例如网络连接、数据库连接，关闭打开的文件等），TestCase 类给我们提供了 setUp 方法和 tearDown 方法，setUp 方法的内容在测试你编写的 TestCase 子类的每个 testXxxx 方法之前都会运 行，而 tearDown 方法的内容在每个 testXxxx 方法结束以后都会执行。这个既共享了初始化代码，又消除了各个测试代码之间可能产生的相互影响。 **JUnit 最佳实践**

Martin Fowler 说过："当你试图打印输出一些信息或调试一个表达式时，写一些测试代码来替代那些传统的方法。"一开始，你会发现你总是要创建一些新的 Fixture，而且测试似乎使你的编程速度慢了下来。然而不久之后，你会发现你重复使用相同的 Fixture，而且新的测试通常只涉及添加一个新的测试 方法。

你可能会写许多测试代码，但你很快就会发现你设想出的测试只有一小部分是真正有用的。你所需要的测试是那些会失败的测试，即那些你认为不会失败的测试，或你认为应该失败却成功的测试。

我 们前面提到过测试是一个不会中断的过程。一旦你有了一个测试，你就要一直确保其正常工作，以检验你所加入的新的工作代码。不要每隔几天或最后才运行测试， 每天你都应该运行一

下测试代码。这种投资很小，但可以确保你得到可以信赖的工作代码。你的返工率降低了，你会有更多的时间编写工作代码。
不要认为压力大，就不写测试代码。相反编写测试代码会使你的压力逐渐减轻，应为通过编写测试代码，你对类的行为有了确切的认识。你会更快地编写出有效率地工作代码。

3. **下面是一些具体的编写测试代码的技巧或较好的实践方法：**
1. 不要用 TestCase 的构造函数初始化 Fixture，而要用 setUp()和 tearDown()方法。
2. 不要依赖或假定测试运行的顺序，因为 JUnit 利用 Vector 保存测试方法。所以不同的平台会按不同的顺序从 Vector 中取出测试方法。
3. 避免编写有副作用的 TestCase。例如：如果随后的测试依赖于某些特定的交易数据，就不要提交交易数据。简单的会滚就可以了。
4. 当继承一个测试类时，记得调用父类的 setUp()和 tearDown()方法。
5. 将测试代码和工作代码放在一起，一边同步编译和更新。（使用 Ant 中有支持 junit 的 task.）
6. 测试类和测试方法应该有一致的命名方案。如在工作类名前加上 test 从而形成测试类名。
7. 确保测试与时间无关，不要依赖使用过期的数据进行测试。导致在随后的维护过程中很难重现测试。
8. 如果你编写的软件面向国际市场，编写测试时要考虑国际化的因素。不要仅用母语的 Locale 进行测试。
9. 尽可能地利用 JUnit 提供地 assert/fail 方法以及异常处理的方法，可以使代码更为简洁。

10.测试要尽可能地小，执行速度快。
JUnit 和 ant 结合
cherami 转贴
ant 提供了两个 target ： junit 和 junitreport
运行所有 测试用例 ，并生成 html 格式的报表
具体操作如下：
1.将 junit.jar 放在 ANT_HOME/lib 目录下
2.修改 build.xml ，加入如下 内容：
<property name="report" value="report" /> <target name="junitreport" depends="clean, compile"> <junit printsummary="on" fork="true" haltonfailure="false" failureproperty="tests.failed" showoutput="true"> <classpath refid="myclasspath"/> <formatter type="xml"/> <batchtest todir="${report}"> <fileset dir="${build}"> <include name="**/*Test.*"/> </fileset> </batchtest> </junit> <junitreport todir="${report}"> <fileset dir="${report}"> <include name="TEST-*.xml"/> </fileset> <report format="frames" todir="${report}"/> </junitreport> <fail if="tests.failed"> --- One or more tests failed, check the report for detail... --- </fail> </target>
运行 这个 target ，ant 会运行每个 TestCase
在 report 目录下就有了 很多 TEST*.xml 和 一些网页
打开 report 目录下的 index.html 就可以看到很直观的测试运行报告，一目了

十五、 XML Interview

1. XML 文档定义有几种形式？它们之间有何本质区别？解析 XML 文档有哪几种方式？

a: 两种形式 dtd（Document Type Definition）and schema。
b: 本质*区别*:**schema** 本身是 xml 的，可以被 XML 解析器解析(这也是从 DTD 上发展 schema 的根本目的)，c:有 DOM,SAX,STAX 等

- **DOM**:处理大型文件时其性能下降的非常厉害。这个问题是由 DOM 的树结构所造成的，这种结构占用的内存较多，而且 DOM 必须在解析文件之前把整个文档装入内存,适合对 XML 的随机访问
- **SAX**:不同于 DOM,SAX 是事件驱动型的 XML 解析方式。它顺序读取 XML 文件，不需要一次全部装载整个文件。当遇到像文件开头，文档结束，或者标签开头与标签结束时，它会触发一个事件，用户通过在其回调事件中写入处理代码来处理 XML 文件，适合对 XML 的顺序访问
- **STAX**:Streaming API for XML (StAX)

十六、 JSP Interview Questions & Answers

1. What is a JSP Page?

A JSP page is a text document that contains two types of text: static data, which can be expressed in any text-based format (such as HTML,WML,XML,etc), and JSP elements, which construct dynamic content.JSP is a technology that lets you mix static content with dynamically-generated content.

2. What is meant by implicit objects? And what are they?

Implicit objects are those objects which are available by default. These objects are instances of classes defined by the JSP specification. These objects could be used within the jsp page without being declared.
The following are the implicit jsp objects:
1. application
2. page
3. request
4. response
5. session
6. exception
7. out
8. config
9. pageContext

request 表示 HttpServletRequest 对象。它包含了有关浏览器请求的信息，并且提供了几个用于获取 cookie, header, 和 session 数据的有用的方法。
response 表示 HttpServletResponse 对象，并提供了几个用于设置送回 浏览器的响应的方法（如 cookies,头信息等）
out 对象是 javax.jsp.JspWriter 的一个实例，并提供了几个方法使你能用于向浏览器回送输出结果。

pageContext 表示一个 javax.servlet.jsp.PageContext 对象。它是用于方便存取各种范围的名字空间、servlet 相关的对象的 API，并且包装了通用的 servlet 相关功能的方法。
session 表示一个请求的 javax.servlet.http.HttpSession 对象。Session 可以存贮用户的状态信息
applicaton 表示一个 javax.servle.ServletContext 对象。这有助于查找有关 servlet 引擎和 servlet 环境的信息
config 表示一个 javax.servlet.ServletConfig 对象。该对象用于存取 servlet 实例的初始化参数。
page 表示从该页面产生的一个 servlet 实例

3. What is the difference between forward and sendRedirect?

Both requestDispatcher.forward() and response.sendRedirect() is used to redirect to new url. Forward is an internal redirection of user request within the web container to a new URL without the knowledge of the user(browser). The request object and the http headers remain intact. sendRedirect is normally an external redirection of user request outside the web container. sendRedirect sends response header back to the browser with the new URL. The browser send the request to the new URL with fresh http headers. sendRedirect is slower than forward because it involves extra server call.

4. What's the Difference between Forward and Include?

- The <jsp:**forward**> action enables you to forward the request to a static HTML file, a servlet, or another JSP. <jsp:forward page="url" /> The JSP that contains the <jsp:**forward**> action stops processing, clears its buffer, and **forwards** the request to the target resource. **Note** that the calling JSP should not write anything to the response prior to the <jsp:**forward**> action. 是页面跳转，就是从一个页面跳转到另外一个页面。
- To "**include**" another resource with a JSP, you have **two options:** the **include** directive and the include action. The **include** directive executes when the JSP is **compiled**, which parses any JSP elements in the included file for a static result that is the same for every instance of that JSP. The syntax for the **include** directive is <@ include file="some-filename" %>. The include action, on the other hand, **executes** for **each** client request of the JSP, which means the file is **not parsed** but **included** in place. This provides the capability to **dynamically change** not only the content that you want to include, but also the output of that content. The syntax for the include action is <jsp:include page="some-filename" flush="true" />. **Note** that the flush attribute must always be included (in JSP 1.1) to force a flush of the buffer in the output stream. 是动态包含，就是在一个主页面中包含另外一个页面，是包含。

5. How does JSP handle run-time exceptions?

You can use the **errorPage** attribute of the page directive to have uncaught runtime exceptions automatically forwarded to an error processing page

6. What are the implicit objects?

Implicit objects are objects that are **created** by the web **container** and contain information **related** to a **particular request**, page, or application. They are:
Request --response --pageContext --session --application --out --config --page --exception

十七、 Applets Interview Questions & Answers

1. **What is an Applet?**

Applet is a java program which is included in a html page and executes in java enabled client browser. Applets are used for creating dynamic and interactive web applications.

2. **Explain the life cycle of an Applet?**

The following methods implement the life cycle of an Applet: init : To initialize the applet each time it's loaded (or reloaded). start : To start the applet's execution, such as when the applet's loaded or when the user revisits a page that contains the applet.stop : To stop the applet's execution, such as when the user leaves the applet's page or quits the browser. destroy : To perform a final cleanup in preparation for unloading.

3. **What happens when an applet is loaded?**

The following sequence happens when an applet is loaded: a) An instance of the applet's controlling class (an Applet subclass) is created. b) The applet initializes itself. c) The applet starts running.

4. **How to make my class as an applet?**

Your class must directly or indirectly extend either Applet or JApplet.

5. **What is AppletContext?**

AppletContext is an interface which provides information about applet's environment.

6. **What is the difference between an Applet and a Java Application?**

The following are the major differences between an Applet and an Application:a. Applets execute within a java enabled browser but an application is a standalone Java program outside of a browser. Both require a JVM (Java Virtual Machine). b. Application requires main() method to trigger execution but applets don't need a main() method. Applets use the init(),start(),stop() and destroy() methods for their life cycle. c. Applets typically use a fairly restrictive security policy. In a standalone application, however, the security policy is usually more relaxed.

7. **What happens when a user leaves and returns to an applet's page?**

When the user leaves the page, the browser stops the applet and when the user returns to the page, the browser starts the applet and normally most browser's dont initialize the applet again..

8. **What are the restrictions imposed on an applet?**

Due to Security reasons, the following restriction are imposed on applets: a. An applet cannot load libraries or define native methods. b. It cannot ordinarily read or write files on the host that's executing it. c. It cannot make network connections except to the host that it came from. d. It cannot start any program on the host that's executing it. e. It cannot read certain system properties.

9. **What are untrusted applets?**

By default, all downloaded applets are untrusted. Untrusted Applets are those applets which cannot access or execute local system files.

10. **How can an applet open a network connection to a computer on the internet?**

Applets are not allowed to open network connections to any computer,except for the host that provided the .class files. This is either the host where the html page came from, or the host specified in the codebase parameter in the applet tag, with codebase taking precendence.

11. Can an applet start another program on the client?

No, applets loaded over the net are not allowed to start programs on the client. That is, an applet that you visit can't start some rogue process on your PC. In UNIX terminology, applets are not allowed to exec or fork processes. In particular, this means that applets can't invoke some program to list the contents of your file system, and it means that applets can't invoke System.exit() in an attempt to kill your web browser. Applets are also not allowed to manipulate threads outside the applet's own thread group.

12. What is the difference between applets loaded over the net and applets loaded via the file system?

There are two different ways that applets are loaded by a Java system. The way an applet enters the system affects what it is allowed to do. If an applet is loaded over the net, then it is loaded by the applet classloader, and is subject to the restrictions enforced by the applet security manager. If an applet resides on the client's local disk, and in a directory that is on the client's CLASSPATH, then it is loaded by the file system loader. The most important differences are :a. applets loaded via the file system are allowed to read and write files b. applets loaded via the file system are allowed to load libraries on the clientc. applets loaded via the file system are allowed to exec processes d. applets loaded via the file system are allowed to exit the virtual machinee. applets loaded via the file system are not passed through the byte code verifier

13. What is the applet class loader, and what does it provide?

Applets loaded over the net are loaded by the applet class loader. For example, the appletviewer's applet class loader is implemented by the class sun.applet.AppletClassLoader. The class loader enforces the Java name space hierarchy. The class loader guarantees that a unique namespace exists for classes that come from the local file system, and that a unique namespace exists for each network source. When a browser loads an applet over the net, that applet's classes are placed in a private namespace associated with the applet's origin. Thus, applets loaded from different network sources are partitioned from each other. Also, classes loaded by the class loader are passed through the verifier. The verifier checks that the class file conforms to the Java language specification - it doesn't assume that the class file was produced by a "friendly" or "trusted" compiler. On the contrary, it checks the class file for purposeful violations of the language type rules and name space restrictions. The verifier ensures that : a. There are no stack overflows or underflows. b. All register accesses and stores are valid. c. The parameters to all bytecode instructions are correct. d. There is no illegal data conversion. e. The verifier accomplishes that by doing a data-flow analysis of the bytecode instruction stream, along with checking the class file format, object signatures, and special analysis of finally clauses that are used for Java

14. What is the applet security manager, and what does it provide?

The applet security manager is the Java mechanism for enforcing the applet restrictions described above. The appletviewer's applet security manager is implemented by sun.applet.AppletSecurity. A browser may only have one security manager. The security manager is established at startup, and it cannot thereafter be replaced, overloaded, overridden, or extended. Applets cannot create or reference their own security manager.

15. If other languages are compiled to Java bytecodes, how does that affect the applet security model?

The verifier is independent of Sun's reference implementation of the Java compiler and the high-level specification of the Java language. It verifies bytecodes generated by other Java compilers. It also verifies bytecodes generated by compiling other languages into the bytecode format. Bytecodes imported over the net that pass the verifier can be trusted to run on the Java virtual machine. In order to pass the verifier, bytecodes have to conform to the strict typing, the object signatures, the class file format, and the predictability of the runtime stack that are all defined by the Java language implementation.

十八、 RMI Interview Questions & Answers

1. What is RMI?

Remote Method Invocation (RMI) is the process of activating a method on a remotely running object. RMI offers location transparency in the sense that it gives the feel that a method is executed on a locally running object.

2. What is the basic principle of RMI architecture?

The RMI architecture is based on one important principle: RMI allows the code that defines the behavior and the code that implements the behavior to remain separate and to run on separate JVMs.

3. What are the layers of RMI Architecture?

The RMI is built on three layers.
a. Stub and Skeleton layerThis layer lies just beneath the view of the developer. This layer intercepts method calls made by the client to the interface reference variable and redirects these calls to a remote RMI Service.
b. Remote Reference Layer. This layer understands how to interpret and manage references made from clients to the remote service objects. The connection is a one-to-one (unicast) link.
c. Transport layer This layer is based on TCP/IP connections between machines in a network. It provides basic connectivity, as well as some firewall penetration strategies.

4. sWhat is the role of Remote Interface in RMI?

The Remote interface serves to identify interfaces whose methods may be invoked from a non-local virtual machine. Any object that is a remote object must directly or indirectly implement this interface. Methods that are to be invoked remotely must be identified in Remote Interface. All Remote methods should throw RemoteException.

5. What is meant by binding in RMI?

Binding is a process of associating or registering a name for a remote object that can be used at a later time to look up that remote object. A remote object can be associated with a name using the Namingclass's bind or rebind methods.

6. What is the difference between using bind() and rebind() methods of Naming Class?

bind method(String name) binds the specified name to a remote object while rebind(String name) method rebinds the specified name to a new remote object,any existing binding for the name is replaced.

7. What are the steps involved to make work a RMI program?

a. Compile the source files.
b. Generate the stubs using rmic.
c. Start the rmiregistry.
d. Start the RMIServer.
e. Run the client program.

8. What is the role of stub in RMI?

A stub for a remote object acts as a client's local representative orproxy for the remote object. The caller invokes a method onthe local stub which is responsible for carrying out the method all on the remote object. When a stub's method is invoked, it does the following:
a. initiates a connection with the remote JVM containing the remote object.
b. marshals (writes and transmits) the parameters to the remote JVM.
c. waits for the result of the method invocation
d. unmarshals (reads) the return value or exception returned
e. returns the value to the caller.

9. What is a skeleton in RMI?

Skeleton was the server side component of stub. But skeleton has been deprecated from JDK1.2 onwards and its not required anymore.

10. What is the difference between Pass By Value and Pass By Reference?

When an Object is "Passed by Value", it means a copy of the object is passed. So even if changes are made to that Object/datatype, it doesn't affect the original value. When an Object is "Passed by Refernce", it means the object is not passed but a reference of the object is passed. so any changes made in the external method gets reflected in all places.

11. What are marker interfaces?（这个要记住）

A so-called marker interface is a Java interface which doesn't actually define any fields or methods. It is just used to "mark" Java classes which support a certain capability -- the class marks itself as implementing the interface. For example, the java.rmi.Remote and java.lang.Cloneable interface.

12. What is meant by Serialisation and Deserialisation?(这题很重要，serialisation 是什么很重要)

Serialization is a way of "flattening", "pickling" or "freeze-drying" objects so that they can be stored on disk, and later read back and reconstituted, with all the links between objects intact. Deserialisation is the reverse process of converting a object from flattened state to live object.

13. What is the protocol used by RMI?

JRMP(java remote method protocol)

十九、 JSF Interview Questions & Answers

1. Explain about JSF?

JSF server side user interface component framework for Java technology-based web applications. Java Server Faces (JSF) is an industry standard and a framework for building component-based user interfaces for web applications.JSF contains an API for representing UI components and managing their state; handling events, server-side validation, and data conversion(转变); defining page navigation; supporting internationalization and accessibility; and providing extensibility for all these features.

2. What are the advantages of JSF? JSF 的好处，有点

The major benefits of JavaServer Faces technology are:
Java Server Faces architecture makes it easy for the developers to use. In JavaServer Faces technology, user interfaces can be created easily with its built-in UI component library, which handles most of the complexities of user interface management.
Offers a clean separation between behavior and presentation.
Provides a rich architecture for managing component state, processing component data, validating user input, and handling events.
Robust event handling mechanism.
Events easily tied to server-side code.
Render kit support for different clients
Component-level control over statefulness
Highly pluggable - components, view handler, etc
JSF also supports internationalization and accessibility
Offers multiple, standardized vendor implementations

3. What are differences between struts and JSF?

In a nutshell, Faces has the following advantages over Struts:
Eliminated the need for a Form Bean
Eliminated the need for a DTO Class
Allows the use of the same POJO on all Tiers because of the Backing Bean
The primary advantages of Struts as compared to JavaServer Faces technology are as follows:

Because Struts is a web application framework, it has a more sophisticated controller architecture than does JavaServer Faces technology. It is more sophisticated partly because the application developer can access the controller by creating an Action object that can integrate with the controller, whereas JavaServer Faces technology does not allow access to the controller. In addition, the Struts controller can do things like access control on each Action based on user roles. This functionality is not provided by JavaServer Faces technology.
Struts includes a powerful layout management framework, called Tiles, which allows you to create templates that you can reuse across multiple pages, thus enabling you to establish an overall look-and-feel for an application.
The Struts validation framework includes a larger set of standard validators, which automatically generate both server-side and client-side validation code based on a set of rules in a configuration file. You can also create custom validators and easily include them in your application by adding definitions of them in your configuration file.

The greatest advantage that JavaServer Faces technology has over Struts is its flexible, extensible UI component model, which includes:[list]

A standard component API for specifying the state and behavior of a wide range of components, including simple components, such as input fields, and more complex components, such as scrollable data tables. Developers can also create their own components based on these APIs, and many third parties have already done so and have made their component libraries publicly av

4. What are the available implementations of JavaServer Faces?

The main implementations of Java Server Faces are:
Reference Implementation (RI) by Sun Microsystems.
Apache MyFaces is an open source JavaServer Faces (JSF) implementation or run-time.
ADF Faces is Oracle's implementation for the JSF standard.

5. What does a typical JSF application consists of?

A typical JSF application consists of the following parts:
JavaBeans components for managing application state and behavior.
Event-driven development (via listeners as in traditional GUI development).
Pages that represent MVC-style views; pages reference view roots via the JSF component tree.

6. What Is a JavaServer Faces Application?

JavaServer Faces applications are just like any other Java web application. They run in a servlet container, and they typically contain the following:

JavaBeans components containing application-specific functionality and data.
Event listeners.
Pages, such as JSP pages.
Server-side helper classes, such as database access beans.
In addition to these items, a JavaServer Faces application also has:

A custom tag library for rendering UI components on a page.
A custom tag library for representing event handlers, validators, and other actions.
UI components represented as stateful objects on the server.
Backing beans, which define properties and functions for UI components.
Validators, converters, event listeners, and event handlers.
An application configuration resource file for configuring application resources.

7. What is Managed Bean?

JavaBean objects managed by a JSF implementation are called managed beans. A managed bean describes how a bean is created and managed. It has nothing to do with the beans functionalities.

8. What is Backing Bean?

Backing beans are JavaBeans components associated with UI components used in a page. Backing-bean management separates the definition of UI component objects from objects that perform application-specific processing and hold data.The backing bean defines properties and handling-logics associated with the UI components used on the page. Each backing-bean property is bound to either a component instance or its value. A backing bean also defines a set of methods that perform functions for the component, such as validating the components data, handling events that the component fires and performing processing associated with navigation when the component activates.

9. What is the different between getRequestParameterMap() and getRequestParameterValuesMap()?

getRequestParameterValuesMap() method is similar to getRequestParameterMap(), but it returns multiple values for the parameters with the same name. This method is useful when using components such as &lt;h:selectMany&gt;

10. Explain JSF Architecture?

JSF was developed using MVC (a.k.a Model View Controller) design pattern so that applications can be scaled(刮去) better with greater maintainability. It is driven by Java Community Process (JCP) and has become a standard. The advantage of JSF is that its both a Java Web user a interface and a framework that fits well with the MVC. It provides clean separation between presentation and behavior. UI (a.k.a User Interface) can be created by page author using reusable UI components and business logic part can be implemented using managed beans.

二十、 Ajax Interview Question

1. What is AJAX? （异步）

AJAX stands for Asynchronous JavaScript And XML. AJAX is not a new programming technology but a concept which makes use of existing technologies. AJAX is based on JavaScript, XML and HTTP requests.

2. Is AJAX a client side technology or a server side technology?

AJAX is a browser based client side scripting technology.

3. How is AJAX different from other browser based technologies?

AJAX allows to refresh smaller portion of a webpage without reloading the entire webpage. This makes an AJAX application faster and more interactive. AJAX uses asynchronous data transfer (HTTP requests) between the browser and the web server, allowing web pages to request small bits of information from the server instead of whole pages.

4. How does AJAX acheive partial page refreshals?

AJAX uses XMLHttpRequest object of Javascript to communicate asynchronously with webserver.

5. Which programming concepts are used by AJAX?

Javascript,XML,HTML and CSS.

6. How will you send data to server using AJAX?

open() and send() methods are used to send data to a web server.open() method takes 3 arguments:
First argument specifies GET or POST method.
Second argument sepcifies the url to be invoked.
Third argument specifies if the request is asynchronous are not.send() method sends the request to the server.

7. **Should I use an HTTP GET or POST for my AJAX calls?**

AJAX requests should use an HTTP GET request when retrieving data where the data will not change for a given request URL. An HTTP POST should be used when state is updated on the server. This is in line with HTTP idempotency recommendations and is highly recommended for a consistent web application architecture.

8. **How to decide on a synchronous request versus an asynchronous request?**

AJAX is basically an asyncronous processing concept. A synchronous request would block in page event processing and there are not many cases where a synchronous request is required.

9. **Explain about onreadystatechange property of XMLHttpRequest?**

onreadystatechange event listening property of XMLHttpRequest which is mapped to a user defined function. The mapped function will be automatically processed after the server response event.

10. **Explain about readyState property of XMLHttpRequest?**

The readyState is a read only property which holds the status of the server' s response. Each time the readyState changes, the onreadystatechange function will be executed.

11. **Explain about responseText property of XMLHttpRequest?**

The responseText property holds the text returned from the web server for the specific request.

12. **Explain about responseXML property of XMLHttpRequest?**

While responseText returns the HTTP response as a string, responseXML returns the response as XML. The responseXML property returns an XML document object, which can be examined and parsed.

13. **How to abort（终止） a XMLHttpRequest?**

Call the abort() method on the request object.

14. **Can i have multiple AJAX requests to the server? If so, how to handle simultaneous requests?**

You can make more than one simultaneous AJAX request. The response may not in the same order as request sequence. Internet Explorer allows only two requests at a time, others browsers may allow up to 5 simultaneous requests. （看浏览器）

15. **Are there any security issues with AJAX?**

AJAX is mostly driven by Javascript and Javascript is loosely built based on Java conventions. JavaScript can not access the local filesystem without the user's permission. An AJAX interaction can only be made with the servers-side component from which the page was loaded, etc makes AJAX pretty safe.

16. **What are the different AJAX based frameworks?**

The prominant AJAX frameworks (actually Javascript libraries) are DWR, Dojo toolkit, Prototype, GWT and Yahoo UI.

17. **Explain about DWR?**

DWR stands for Direct Web Reporting. DWR allows Javascript in a browser to interact with Java on a server and helps you manipulate web pages with the results. On the server side DWR uses a Servlet to interact with the Java objects and returns object representations of the Java objects or XML documents. DWR is also sometimes referred as Reverse-Ajax.

18. Explain about Dojo Toolkit?

The following are the features of Dojo Toolkit:
- Powerful AJAX-based I/O abstraction (remoting).
- Graceful degradation.
- Backward, forward, bookmarking support.
- Aspect-oriented event system.
- Markup-based UI construction through widgets.
- Widget prototyping.
- Animation

19. Explain about Prototype?

Prototype JavaScript library contains a set of JavaScript objects for representing AJAX requests and contains utility functions for accessing in page components and DOM manipulations.

20. Explain about GWT?

With Google Web Toolkit (GWT), you write your AJAX front-end in the Java programming language which GWT then cross-compiles into optimized JavaScript that automatically works across all major browsers.

二十一、 JMS Interview

1. What is Messaging?

Messaging is a method of **communication** between **software** components or **applications**

2. What is JMS?

Java Message Service is a Java API that allows **applications** to create, send, receive, and read messages.

3. What are the features of JMS?

The following are the important features of JMS:
- **Asynchronous** Processing.
- Store and forwarding.
- **Guaranteed** Delivery.
- Provides **location** transparency.
- Service based **Architecture**.

4. What is poison messages? And how to handle poison messages?

Poison messages, messages the application can never successfully process. A badly-formatted message arrives on a queue. Such a message might **make** the receiving **application fail** and back out the receipt of the message. In this situation, such a message might be received, then returned to the queue, repeatedly. These messages are known as **poison messages**. The **ConnectionConsumer** must be able to **detect poison messages** and reroute them to an alternative destination.

5. **What are the steps to send and receive JMS message ?**

1) lookup the **ConnectionFactory** : A **connection factory** is the object a client uses to create a connection to a provider. A **connection factory** encapsulates a set of connection configuration parameters that has been defined by an administrator. Each connection factory is an instance of the ConnectionFactory, QueueConnectionFactory, or TopicConnectionFactory interface
2) **lookup the destination:** A destination is the object a client uses to specify the **target** of **messages** it produces and the source of messages it consumes.
3) **Create** the **Connection:** Connection connection = connectionFactory.createConnection();
4) Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
5) Create **Message Producers:** You use a Session to create a MessageProducer for a destination.
6) **Create** message and **send** message
7) **create** Message **Consumers** and **receive** message

6. **What are two messaging models or messaging domains?**

The **two** messaging models are :
Point-to-Point Messaging domain.
Publish/Subscribe Messaging domain

7. Explain Point-to-Point Messaging model

A **point-to-point (PTP)** product or application is built around the concept of message queues, senders, and receivers. **Each** message is addressed to a **specific** queue, and **receiving** clients extract messages from the queue(s) established to hold their messages. **Queues** retain all messages sent to them **until** the messages are consumed or until the messages expire. **Point-to-Point** Messaging has the following **characteristics:**
Each Message has only one consumer.
The receiver can fetch the message whether or not it was running when the client sent the message.
The receiver acknowledges the successful processing of a message.

8. Explain Pub/Sub Messaging model.

In a **publish/subscribe (pub/sub)** product or application, clients address messages to a topic. **Publishers** and **subscribers** are generally anonymous and may dynamically publish or subscribe to the content hierarchy. The system takes care of distributing the messages arriving from a topic's multiple publishers to its multiple subscribers. Topics retain messages only as long as it takes to distribute them to current subscribers. **Pub**/**sub** messaging has the following **characteristics**:
Each message may have multiple consumers.
Publishers and subscribers have a timing dependency. A client that subscribes to a topic can consume only messages published after the client has created a subscription, and the subscriber must continue to be active in order for it to consume messages.

9. What are the two types of Message Consumption?（消耗）

Synchronous Consumption :A **subscriber** or a receiver explicitly fetches the message from the destination by calling the receive method. The receive method can block until a message arrives or can time out if a message does not arrive within a specified time limit.
Asynchronous Consumption : A client can **register** a **message listener** with a consumer. A message listener is similar to an event listener. Whenever a message arrives at the destination, the JMS provider delivers the message by calling the listener's **onMessage** method, which acts on the contents of the message.

10. What is a connection factory?

A **connection factory** is the object a client uses to create a connection with a provider. A connection factory encapsulates a set of connection configuration parameters that has been defined by an administrator.Each connection factory is an instance of either the QueueConnectionFactory or the TopicConnectionFactory interface.

11. What is a destination?

A destination is the object a client uses to specify the **target** of messages it produces and the source of messages it consumes. In the PTP messaging domain, destinations are called queues and in the pub/sub messaging domain, destinations are called topics.

12. What is a message listener?

A message listener is an object that acts as an asynchronous event handler for messages. This object implements the MessageListener interface, which contains one method, onMessage. In the **onMessage** method, you define the actions to be taken when a message arrives.

13. What is a message selector?

Message selector **filters** the messages received by the consumer based on a criteria. **Message selectors** assign the work of filtering messages **to the JMS provider** rather than to the application. A message selector is a String that contains an expression. The syntax of the expression is based on a subset of the SQL92 conditional expression syntax. The message consumer then receives only messages whose headers and properties match the selector.

14. What are the parts of a JMS message?

A JMS message has **three parts:**
header ;Properties (optional) ; body (optional)

15. What is a message header?

A JMS message header contains a number of predefined fields that contain values that both clients and providers use to identify and to route messages. Each header field has **associated setter** and **getter** methods, which are documented in the description of the Message interface.

16. Name few message headers which are automatically assigned during message creation?

Following headers are automatically assigned:
JMSDestination; JMSDeliveryMode; JMSMessageID; JMSTimestamp; JMSExpiration; JMSRedelivered; JMSPriority

17. Name few subclasses of JMSException.

MessageFormatException; MessageEOFException; InvalidClientIDException; InvalidDestinationException; InvalidSelectorException

18. How many types of Messages are there in JMS and What are they?

There are **5 types** of Messages. They are:
TextMessage : A java.lang.String object (for example, the contents of an Extensible Markup Language file).
MapMessage : A set of name/value pairs, with names as String objects and values as primitive types in the Java programming language. The entries can be accessed sequentially by enumerator or randomly by name. The order of the entries is undefined.
BytesMessage : A stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format.
StreamMessage: A stream of primitive values in the Java programming language, filled and read sequentially.
ObjectMessage: A Serializable object in the Java programming language.

19. Explain about JMS Client, Producer, Consumer, Provider.

Java applications that use JMS are called **JMS clients**, and the messaging system that handles routing and delivery of messages is called the **JMS provider**. A JMS client that sends a message is called a **producer,** while a JMS client that receives a message is called a **consumer.** A single JMS client can be both a producer and a consumer.

20. What is a Message Driven Bean?

Message-driven beans (MDBs) are stateless, server-side, transaction-aware components for processing asynchronous JMS messages. MDBs were introduced as part of EJB 2 specification. Message-driven beans process messages delivered via the Java Message Service. A message-driven bean is an enterprise bean that allows J2EE applications to process messages asynchronously. It acts as a JMS message listener, which is similar to an event listener except that it receives messages instead of events. The messages may be sent by any J2EE component--an application client, another enterprise bean, or a Web component or by a JMS application or system that does not use J2EE technology.

21. Why do MDB's dont have component interfaces or JNDI?

MDB's are **only for internal processing**. You **cannot** look up a MDB from **client**. If you want to access MDB you rather send a message to the Queue. Hence MDB's doesnt have a JNDI name. And for the same reason they dont have component interfaces.

22. How are MDBs different from Session and Entity Beans?

The most visible difference between message-driven beans and session and entity beans is that **clients do not access message-driven beans through interfaces**. Unlike a session or entity bean, a message-driven bean **has only a bean class.**

What is the difference between Message Driven Beans and Stateless Session beans? 可以看做是 MDB 和 stateless session 的区别

In **several ways**, the dynamic creation and allocation of **message-driven** bean instances **like** the behavior of **stateless session** EJB instances, which exist only for the duration of a particular method call.

However, **message-driven** beans are **different** from **stateless session** EJBs (and other types of EJBs) in **several significant ways**:

Message-driven beans process **multiple JMS** messages **asynchronously**, rather than processing a serialized sequence of method calls.

Message-driven beans have no home or remote interface, and therefore cannot be directly accessed by internal or external clients. **Clients** interact with message-driven beans **only** indirectly, by sending a message to a JMS Queue or Topic.

Note: **Only** the **container** directly interacts with a **message-driven** bean by creating bean instances and passing JMS messages to those instances as necessary.

The **Container** maintains the **entire** lifecycle of a **message-driven** bean; instances **cannot** be created or removed as a result of client requests or other API calls

created or removed as a result of client requests or other API callsWhat are the two parts of a message and explain them?

A message basically has **two parts** : a **header** and **payload**(负载). The **header** is comprised of special fields that are used to identify the message, declare attributes of the message, and provide information for routing. **Payload** is the type of application data the message contains.

23. What are the two types of delivery modes?

JMS delivery mode are of **two types** Non-Persistent and Persistent. **Non-Persistent** messages do not need to be stored in persistent storage in case of client failure. A JMS provider must deliver this kind of message at-most-once, i.e., the message can be lost, but can only be delivered once. **Persistent messages** are stored in persistent storage to be delivered at a later date if a client is unavailable. A JMS provider must deliver this kind of message once-and-only once, i.e., it cannot be lost and cannot be delivered more than once.

What is the different between JMS and RPC?

RPC : **Remote procedure Call**-----In RPC the method **invoker waits** for the method to finish execution and **return** the **control** back to the invoker. Thus it is completely **synchronous** in nature.

JMS : **Java Messaging System**----- While in JMS the message sender just **sends** the message to the **destination** and continues it's own **processing**. The sender does **not wait** for the receiver to **respond**. This is **asynchronous** behavior.