that this default mode also allows for annotation-driven autowiring, if activated. "no" refers to externally driven autowiring only, not affecting any autowiring demands that the bean class itself expresses.
    - "byName" Autowiring by property name. If a bean of class Cat exposes a "dog" property, Spring will try to set this to the value of the bean "dog" in the current container. If there is no matching bean by name, nothing special happens.
    - “byType" Autowiring if there is exactly one bean of the property type in the container. If there is more than one, a fatal error is raised, and you cannot use byType autowiring for that bean. If there is none, nothing special happens.
    - "constructor" Analogous to "byType" for constructor arguments. If there is not exactly one bean of the constructor argument type in the bean factory, a fatal error is raised. Note that explicit dependencies, i.e. "property" and constructor-arg" elements, always override autowiring. Note: This attribute will not be inherited by child bean definitions. Hence, it needs to be specified per concrete bean definition.
- Spring transaction propagation
    - **REQUIRED**:It means that the same transaction will be used if there is an already opened transaction in the current bean method execution context. If there is no existing transaction, the Spring container will create a new one. It is a default behavior of propagation.
    - **REQUIRES_NEW**:It means that a new physical transaction will always be created by the container. In other words, the inner transaction may commit or rollback independently of the outer transaction.
    - **NESTED**:It makes nested Spring transactions to use the same physical transaction but sets save points between nested invocations. So inner transactions may also rollback independently of outer transaction. This behavior should only be used with Spring JDBC managed transactions.
    - **MANDATORY**:It states that an existing opened transaction must already exist. If not an exception will be thrown by the container.
    - **NEVER**:It states that an existing opened transaction must not already exist. If a transaction exists an exception will be thrown by the container.
    - **NOT_SUPPORTED**: It will execute outside of the scope of any transaction. If an opened transaction already exists, it will be paused.
    - **SUPPORTS**: It will execute in the scope of a transaction if an opened transaction already exists. If there isn’t any already opened transaction the method will execute anyway but is a non-transactional way

Pro and Cons of Stored procedure
Disadvantages:

- **Limited Coding Functionality**
    - **Stored procedure code is not as robust as app code, particularly in the area of looping (not to mention that iterative constructs, like cursors, are slow and processor intensive)**

- **Portability**
    - **Complex Stored Procedures that utilize complex, core functionality of the RDBMS used for their creation will not always port to upgraded versions of the same database. This is especially true if moving from one database type (Oracle) to another (MS SQL Server).**
- **Testing**
    - **Any data errors in handling Stored Procedures are not generated until runtime**
- **Cost**
    - **Depending on your corporate structure and separation of concern for development, there is the potential that Stored Procedure development could potentially require a dedicated database developer. Some businesses will not allow developers access to the database at all, requiring instead a separate DBA. This will automatically incur added cost.**
    - **Some companies believe (and sometimes it's true, but not always) that a DBA is more of a SQL expert than an application developer, and therefore will write better Stored Procedures. In that case, an extra developer in the form of a DBA is required.**
- **Location of Business Rules**
    - **Since SP's are not as easily grouped/encapsulated together in single files, this also means that business rules are spread throughout different Stored Procedures. App code architecture helps to ensure that business rules are encapsulated in single objects.**
    - **There is a general opinion that business rules / logic should not be housed in the data tier**

Advantages:

- **Speed / Optimization**
    - **Stored procedures are cached on the server**
    - **Execution plans for the process are easily reviewable without having to run the application**
- **Utilization of Set-based Processing**

    - **The power of SQL is its ability to quickly and efficiently perform set-based processing on large amounts of data; the coding equivalent is usually iterative looping, which is generally much slower**
- **Security**
    - **Limit direct access to tables via defined roles in the database**
    - **Provide an "interface" to the underlying data structure so that all implementation and even the data itself is shielded.**
    - **Securing just the data and the code that accesses it is easier than applying that security within the application code itself**

Types of Index:

Structure wise:

- Bitmap index
  A bitmap index is a special kind of index that stores the bulk of its data as bit arrays (bitmaps) and answers most queries by performing bitwise logical operations on these bitmaps. The most commonly used indexes, such as B+trees, are most efficient if the values they index do not repeat or repeat a smaller number of times. In contrast, the bitmap index is designed for cases where the values of a variable repeat very frequently. For example, the gender field in a customer database usually contains at most three distinct values: male, female or other. For such variables, the bitmap index can have a significant performance advantage over the commonly used trees.
- Dense index[edit]

A dense index in databases is a file with pairs of keys and pointers for every record in the data file. Every key in this file is associated with a particular pointer to a record in the sorted data file. In clustered indices with duplicate keys, the dense index points to the first record with that key.[3]

- Sparse index[edit]

A sparse index in databases is a file with pairs of keys and pointers for every block in the data file. Every key in this file is associated with a particular pointer to the block in the sorted data file. In clustered indices with duplicate keys, the sparse index points to the lowest search key in each block.

- Reverse index[edit]
  Main article: Reverse index
  A reverse key index reverses the key value before entering it in the index. E.g., the value 24538 becomes 83542 in the index. Reversing the key value is particularly useful for indexing data such as sequence numbers, where new key values monotonically increase.

or

- single-column index
- composite index

ClassLoader:

类加载过程：
1、寻找jre目录，寻找jvm.dll，并初始化JVM；

2、产生一个Bootstrap Loader（启动类加载器）；

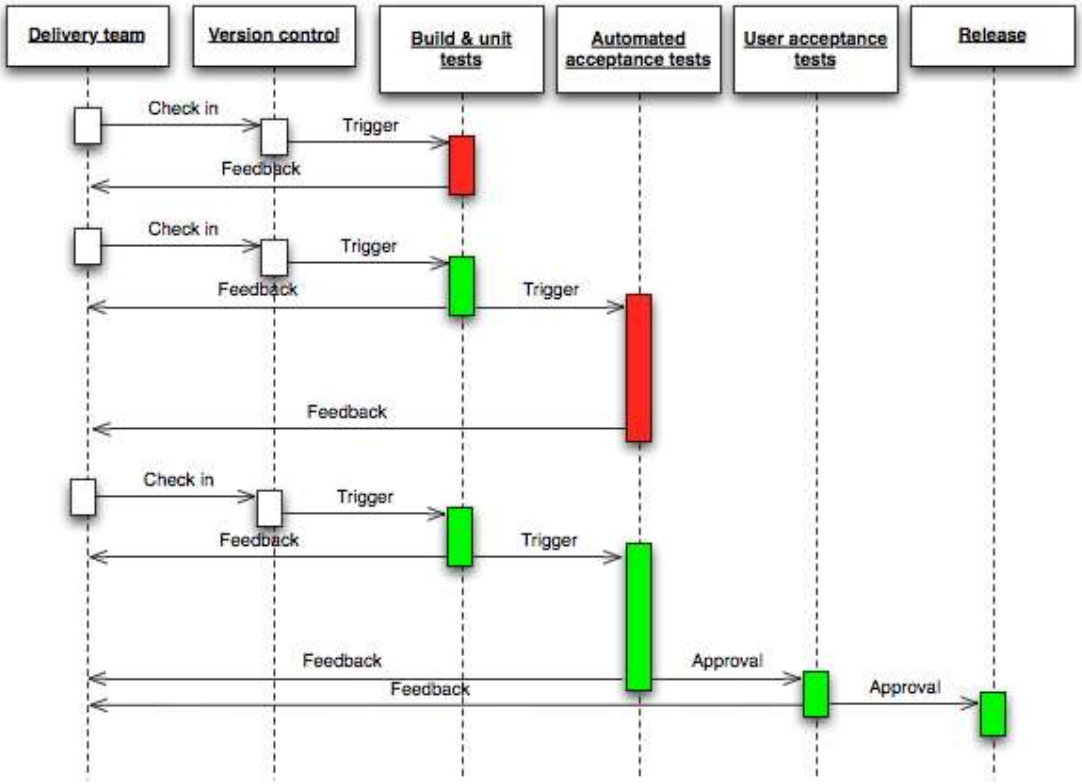
3、Bootstrap Loader自动加载Extended Loader（标准扩展类加载器），并将其父Loader设为Bootstrap Loader。

4、Bootstrap Loader自动加载AppClass Loader（系统类加载器），并将其父Loader设为Extended Loader。

5、最后由AppClass Loader加载HelloWorld类。

Engineer practise

- Continuous delivery
    - Continuous Delivery (CD) is a software engineering approach in which teams keep producing valuable software in short cycles and ensure that the software can be reliably released at any time. It is used in software development to automate and improve the process of software delivery.



- TDD(Test-Driven-Development)
  - Add a test
  - Run all tests and see if the new one fails
  - Write some code
  - Run tests
  - Refactor code
  - Repeat
- cucumber (ref: http://www.guru99.com/cucumber-tutorials.html)
  - You need 2 Files – Features and Step Definition to execute a Cucmber test scenario
  - Features file contain high level description of the test scenario in simple language
  - Steps Definition file contains the actual code to execute the test scenario in the Features file.
  - Feature files
    - Example:
    - Feature: Visit career guide page in career.guru99.com
    - Scenario : Visit career.guru99.com
    - Given: I am on career.guru99.com
    - When: I click on career guide menu
    - Then: I should see career guide page

- Step definition
  - Step 1:
    - Given (/^ I am on career.guru99.com$/) do
    - Browser.goto "http://career.guru99.com" -This will visit career.guru99 on browser
    - end
  - Step 2:
    - When (/^ click on career guide menu$/) do
    - Browser.text (:name, " career guide").click – This will click "career guide menu"
    - end
  - Step 3:
    - Then (/^ I should see career guide page$/) do
    - Browser.goto "http://career.guru99.com/category/career-guide/" - It will visit "career guide page"
    - end
-

Web Service:

- Soap:
  - WSDL:
    - WSDL stands for Web Services Description Language.
    - WSDL is a language for describing web services and how to access them.
    - WSDL is written in XML.
  - XSD:
    - XML Schema is an XML-based alternative to DTD.
    - XML Schemas are much more powerful than DTDs.
    - The XML Schema language is also referred to as XML Schema Definition (XSD).
  - SOAP structure:
    - **Envelope** − Defines the start and the end of the message. It is a mandatory element.
    - **Header** − Contains any optional attributes of the message used in processing the message, either at an intermediary point or at the ultimate end-point. It is an optional element.
    - **Body** − Contains the XML data comprising the message being sent. It is a mandatory element.
    - **Fault** − An optional Fault element that provides information about errors that occur while processing the message.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
```

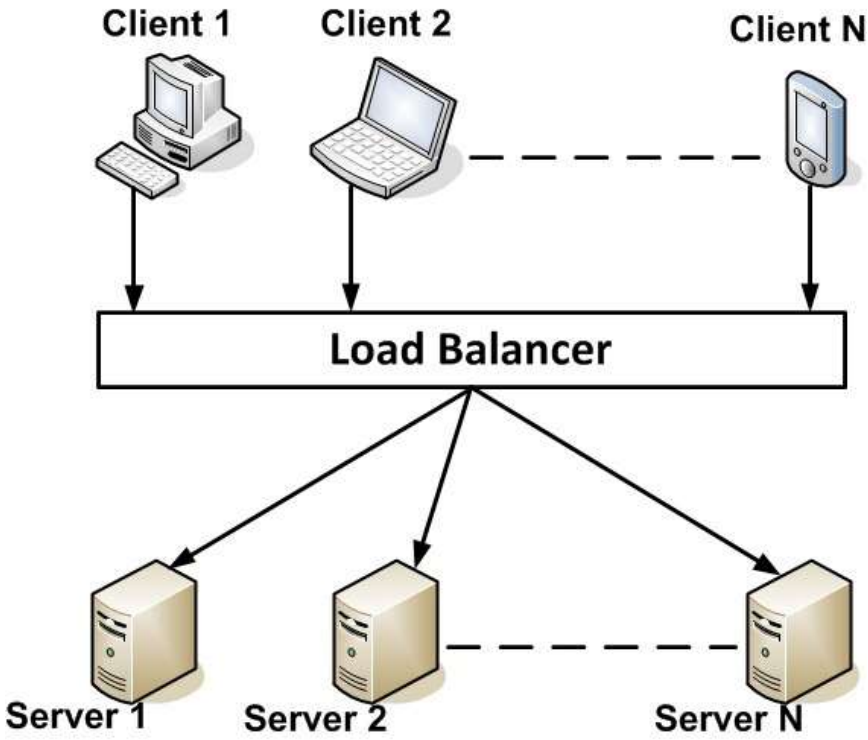
```
<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>IBM</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

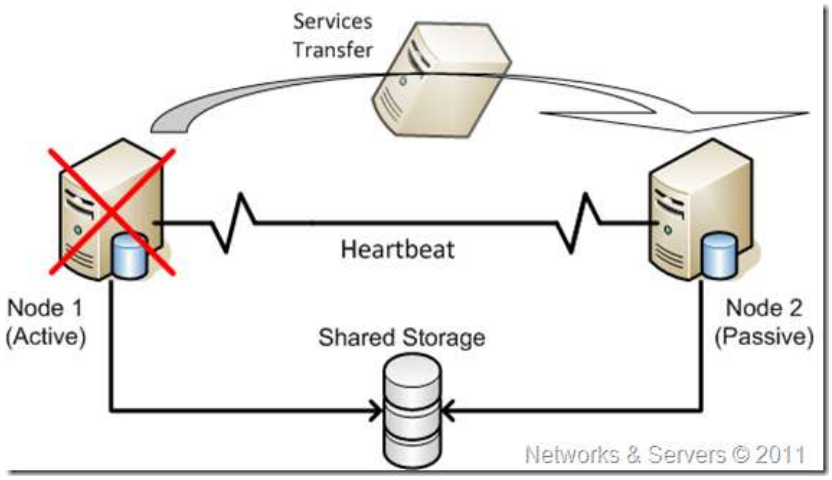
- SOAP vs REST

	SOAP	REST
	HTTP or SMTP	HTTP
	Payload are XML	normally json
	Heavy-weighted (if you write JS code to send SOAP request… you have to create the whole XML structure)	Light-weighted
	Build-in error handling and xsd to validate	has to be done manually
	By default stateless. but it can be made stateful (stateful means it maintains state information between messages calls. SOAP server will send sessionId within the SOAP header)	stateless

-
- How to build a scalable web service (ref:http://www.aosabook.org/en/distsys.html)
    - Availability
        - Load balance:

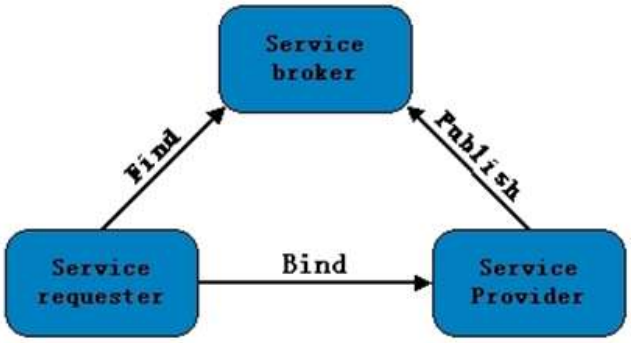


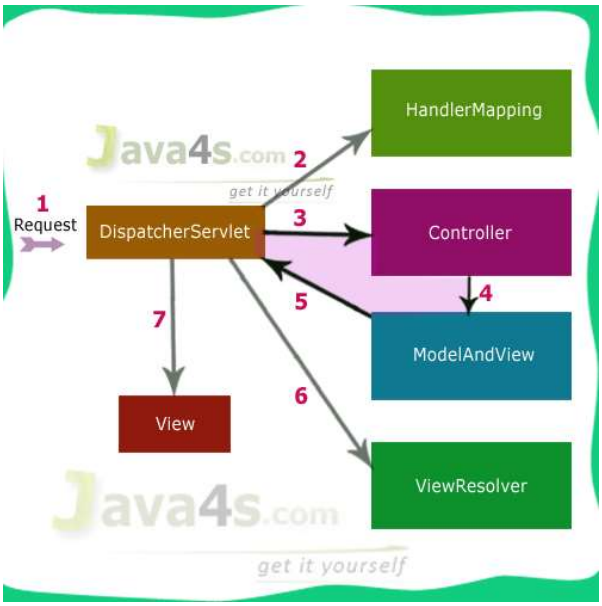
- Failover:



    - Reliability: write a data and can be retrieved in the future; all the request at the same time should return the same data.
    - Performance:
    - Manageablity: Maintainence and upgrade effort
    - Cost
- SOA(Service Oriented architecture)
    - Service provider: publish its service and give response to the request
    - Service broker: register the published services, classify them and provide search engine

- Service consumer: find requested service from service broker and utilise the service



- Spring MVC
  - WorkFlow:
    - Step **1**: First request will be received by DispatcherServlet
    - Step **2**: DispatcherServlet will take the help of HandlerMapping and get to know theController class name associated with the given request
    - Step **3**: So request transfer to the Controller, and then controller will process the request by executing appropriate methods and returns ModeAndView object (contains *Model* data and *View* name) back to the DispatcherServlet
    - Step 4: Now DispatcherServlet send the model object to the ViewResolver to get the actual view page
    - Step **5**: Finally DispatcherServlet will pass the *Model* object to the *View* page to display the result
    - 

Java 7 new features:

- **Reference(http://radar.oreilly.com/2011/09/java7-features.html)**
- **Diamond operator**
    - Map<String, List<Integer>> map =new HashMap<>();
- **Using String in switch statement**
- **Automatic resource management**
- **Numeric literals with underscores**
- New file system API
    - Create a WatchService. This service consists of a queue to hold WatchKeys
    - Register the directory/file you wish to monitor with this WatchService
    - While registering, specify the types of events you wish to receive (create, modify or delete events)
    - You have to start an infinite loop to listen to events
    - When an event occurs, a WatchKey is placed into the queue
    - Consume the WatchKey and invoke queries on it
- New concurrency framework
    - Basically the Fork-Join breaks the task at hand into mini-tasks until the mini-task is simple enough that it can be solved without further breakups. It's like a divide-and-conquer algorithm. One important concept to note in this framework is that ideally no worker thread is idle. They implement a work-stealing algorithm in that idle workers "steal" the work from those workers who are busy.
    - ForkJoinPool → ExecuterService
    - ForkJoinTask → Thread
    - RecursiveAction.compute() → Runnable.run()
    - RecursiveTask.compute() → Callable.call()

DataBase

- Execution plan:

```
SQL> explain plan for SELECT *
  2  FROM PRODUCT, BUYER
  3  WHERE PRODUCT.ID = BUYER.PRODID;

Explained.

SQL> select * from table(dbms_xplan.display);

PLAN_TABLE_OUTPUT
------------------------------------------

------------------------------------
| Id  | Operation           | Name    |
------------------------------------
|   0 | SELECT STATEMENT    |         |
```

```
|   1 |  MERGE JOIN          |          |
|   2 |   SORT JOIN          |          |
|   3 |    TABLE ACCESS FULL| BUYER    |
|*  4 |   SORT JOIN          |          |
|   5 |    TABLE ACCESS FULL| PRODUCT  |
---------------------------------------
```

Execution tree plan are read in postorder. in this case, it is 3 2 5 4 1