## <http://www.javatechblog.com/java/difference-between-soap-and-restful-web-service-in-java/>

## Difference between SOAP and RESTful Web Service in Java

Difference between SOAP and RESTful Web Service is one of the frequently asked [interview questions](http://www.javatechblog.com/tag/java-interview-questions/) in Java. Here is a detailed comparison between them.

|  |  |
| --- | --- |
| **REST** | **SOAP** |
| REST (REpresentational State Transfer) is an architectural style on which RESTFul web services are built. | SOAP (Simple Object Access Protocol) is a protocol. |
| [JAX-RS](https://docs.oracle.com/javaee/6/tutorial/doc/giepu.html) is the specification defined for developing RESTful web services | [JAX-WS](https://docs.oracle.com/javaee/6/tutorial/doc/bnayl.html) is the specification defined for developing SOAP web services. |
| REST permits different data format such as Plain text, HTML, XML, JSON etc. | SOAP permits XML data format only, actual response is bundled inside a SOAP message which is always in XML format. |
| RESTful Web services are heavily dependent upon HTTP protocol. | SOAP messages can be sent over to any transport mechanism e.g. TCP, FTP, SMTP or any other protocol. |
| Processing a RESTful web service request is much faster because REST serves JSON that is faster to parse than XML | Processing a SOAP request is slower because of the overhead in parsing XML markup |
| RESTful web service takes full advantage of the web caching mechanism | SOAP web services totally ignore web caching mechanism. |
| REST is a concept that does not tie with any protocols. SOAP based web services can be implemented in RESTful style | SOAP can’t use REST architecture. |
| URL typically references the resource being accessed/deleted/updated | SOAP uses XML messages to identify the desired web procedure or resource to be invoked. |
| REST is lightweight as compared to SOAP, it doesn’t require CPU consuming XML parsing | SOAP is heavyweight compared to REST because it requires XML parsing and require a SOAP header for every message. |
| No built-in error handling | Built-in error handling (SOAP faults) |
| Security in RESTful web service can be implemented using standard and traditional authentication methods | SOAP defines own security mechanism |
| Not suitable for distributed computing environment | Designed to handle distributed computing environments |

## Why is String Immutable in Java? ([answer](http://java67.blogspot.sg/2014/01/why-string-class-has-made-immutable-or-final-java.html))

## One of my favorite Java interview question. The String is Immutable in java because java designer thought that string will be heavily used and making it immutable allow some optimization easy sharing same String object between multiple clients. See the link for the more detailed answer. This is a great question for Java programmers with less experience as it gives them food for thought, to think about how things works in Java, what Jave designers might have thought when they created String class etc.

## 

## 

## Read more: <http://javarevisited.blogspot.com/2015/10/133-java-interview-questions-answers-from-last-5-years.html#ixzz4YDQZgiiE>

## How HashSet is internally implemented in Java?

Most of us know how HashMap is internally implemented (if you want to know refer my post [How HashMap works](http://www.javatechblog.com/java/hashmap-works-internally-java/)) but many might not know how HashSet is internally implemented in Java. This is actually a frequently asked [java interview question](http://www.javatechblog.com/tag/java-interview-questions/). You might be surprised to know that HashSet internally uses HashMap.

Let us first see what is HashSet and then we discuss about its internal implementation.

## HashSet

[HashSet](https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html) is an implementation of the Set interface. It is a collection that does not allow duplicate elements. HashSet does not guarantee that the iteration order will remain constant over time i.e. it does not guarantee the order of elements. This class permits null element. HashSet is not synchronized and hence not thread safe. HashSet can be synchronized by callingCollections.synchronizedSet(new HashSet(…)); HashSet does not provide index based accessing as in a List and can be accessed using an iterator.

### HashSet – primitive example

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | import java.util.HashSet;  public class HashSetExample {  public static void main(String[] args) {  HashSet<String> nameSet = new HashSet<String>(0);  nameSet.add("Arun");  nameSet.add("Arun");  nameSet.add("Vijay");  nameSet.add("Vijay");  nameSet.add(null);  System.out.println("NameSet size: " + nameSet.size());  System.out.println(nameSet);  HashSet<Integer> numSet = new HashSet<Integer>(0);  numSet.add(1);  numSet.add(2);  numSet.add(3);  numSet.add(3);  System.out.println("NumberSet size: " + numSet.size());  System.out.println(numSet);  }  } |

When you run the above program you will get below output,

NameSet size: 3

[null, Arun, Vijay]

NumberSet size: 3

[1, 2, 3]

As you can see from the above output, HashSet does not contain duplicate elements.

### HashSet – Custom Object example

In order to add custom objects to HashSet, the custom object must override the equals () and hashcode() method properly (Read my another post on [equals () and hashcode()](http://www.javatechblog.com/java/equals-hashcode-java/) to know more about equals and hashcode implementation). Otherwise the HashSet will not work as expected.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84 | import java.util.HashSet;  import java.util.Set;  public class HashSetObjectExample {  public static void main(String[] args) {  Employee e1 = new Employee("Arun", "Kumar", 1);  Employee e2 = new Employee("Vijay", "Kumar", 2);  Set<Employee> empSet = new HashSet<>(0);  empSet.add(e1);  empSet.add(e2);  empSet.add(e2);  System.out.println("Employee Set size: " + empSet.size());  System.out.println(empSet);  }  }  class Employee {  private String firstName;  private String lastName;  private int empid;  public Employee(String firstName, String lastName, int empid) {  super();  this.firstName = firstName;  this.lastName = lastName;  this.empid = empid;  }  public String getFirstName() {  return firstName;  }  public void setFirstName(String firstName) {  this.firstName = firstName;  }  public String getLastName() {  return lastName;  }  public void setLastName(String lastName) {  this.lastName = lastName;  }  public int getEmpid() {  return empid;  }  public void setEmpid(int empid) {  this.empid = empid;  }  @Override  public int hashCode() {  final int prime = 31;  int result = 1;  result = prime \* result + empid;  result = prime \* result  + ((firstName == null) ? 0 : firstName.hashCode());  result = prime \* result  + ((lastName == null) ? 0 : lastName.hashCode());  return result;  }  @Override  public boolean equals(Object obj) {  if (this == obj)  return true;  if (obj == null)  return false  if (getClass() != obj.getClass())  return false;  Employee other = (Employee) obj;  if (empid != other.empid)  return false;  if (firstName == null) {  if (other.firstName != null)  return false;  } else if (!firstName.equals(other.firstName))  return false;  if (lastName == null) {  if (other.lastName != null)  return false;  } else if (!lastName.equals(other.lastName))  return false;  return true;  }  @Override    public String toString() {  return "Employee [firstName=" + firstName + ", lastName=" + lastName    + ", empid=" + empid + "]";  }} |

When you run the above program you will get below output,

Employee Set size: 2

[Employee [firstName=Vijay, lastName=Kumar, empid=2], Employee [firstName=Arun, lastName=Kumar, empid=1]]

If you haven’t overridden equals or hashcode method, the above HashSet would contain duplicate employee object which breaks the HashSet principle. So it is important to understand the working of equals and hashcode method.

## Internal working of HashSet

HashSet is backed by a hash table (actually a HashMap instance). When you create an object of HashSet in Java, it internally creates an instance of HashMap. This can be seen from the HashSet constructors as below,

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public HashSet() {  map = new HashMap<>();  }  public HashSet(int initialCapacity) {  map = new HashMap<>(initialCapacity);  } |

Here map is a private variable in HashSet class defined as below,

private transient HashMap<E,Object> map;

You might know that HashMap does not allow duplicate keys and this property is utilized by HashSet to maintain unique elements. So the HashSet stores the elements as Key in the HashMap and the values for all the keys is an instance of Object class.

This can be seen from the add() method in HashSet.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | /\*  \* @param e element to be added to this set  \*/  public boolean add(E e) { // e is the element to be added  return map.put(e, PRESENT)==null;  } |

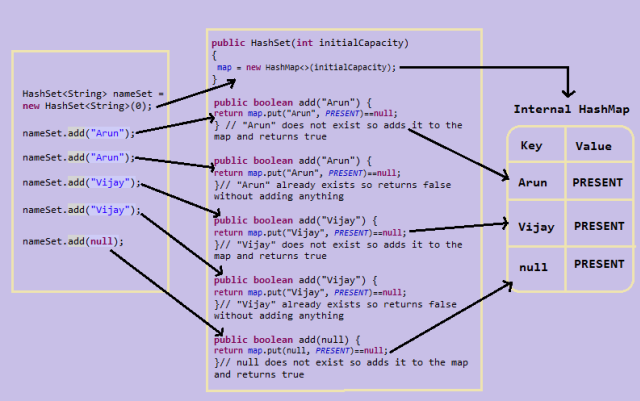
The value *PRESENT* is defined as below,

// Dummy value to associate with an Object in the backing Map

private static final Object *PRESENT* = new Object();

So it is very much clear that whenever add() method is called in HashSet, the elements are stored as a key in the HashMap. It can be seen that PRESENT is defined as a constant hence all the keys will contain the same value in HashMap.

The add() method returns true if the element to be added is not already present in the HashMap and returns false if the element already exists. If the element already exist, the HashSet is unaffected by the add() operation. Here is a pictorial representation of various operations.



So we have seen how HashSet is internally implemented in Java. Don’t think of HashSet as HashMap as it will bring confusion. Just think HashSet as a collection that do not allow duplicate elements. However when adding custom objects, it is important to know the usage of equals and hashcode methods. It is also important to understand how HashMap works internally, in order to explain the internal working of HashSet.

## Frequently asked Java interview questions for experienced – 2 to 5 years

This is a continuation of my previous posts on frequently asked [Java interview questions](http://www.javatechblog.com/tag/java-interview-questions/) for experienced Java developers. Let see some of the important question and answers.

### 1. What is Generics in Java?

This is one of the most important Java interview questions. Especially those who have higher experience in Java should have thorough knowledge on Generics. Refer my previous post about[Generics in Java](http://www.javatechblog.com/java/generics-in-java-with-example-programs/).

## 2. Can we add an element while iterating over a list?

We should not modify collection’s structure once an iterator is defined. We can remove an element from the underlying collection with the iterator’s remove () method after a call to the next () method. This method can be called only once per call to next (). The behavior of an iterator is unspecified if the underlying collection is modified while the iteration is in progress in any way other than by calling this method. It will throw java.util.ConcurrentModificationException if we try to add element while the iteration is in progress.

Throws IllegalStateException – if the next method has not yet been called and the remove method is called or the remove method has already been called after the last call to the next method.

## 3. What is the difference between an application server and a web server?

A web server typically contains a servlet container(web container) to handle HTTP requests whereas an application server have servlet container as integral part of them and serves business logic through any protocol, including, but not limited, to HTTP. That means an Application Server can do whatever Web Server is capable of. In addition an Application Server also has a EJB container, supports distributed transaction and provides application level services like Connection pooling , Transaction management, load balancing, Messaging etc. while Web Server only supports Servlets and JSP.

E.g. Apache Tomcat is a web server and JBoss is an application server.

## 4. Which Collection can be used for holding key value pair in the insertion order?

[LinkedHashMap](https://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashMap.html) can be used for holding key-value pair in the insertion order.

## 5. Why String is Immutable in Java?

Below are the key factors why String is made final or immutable.

* Optimization and performance
* Security
* Multithreading Benefits

Read more about each factor – [5 Reasons why string is immutable.](http://java67.blogspot.in/2014/01/why-string-class-has-made-immutable-or-final-java.html)

## 6. Difference between wait and sleep method in Java.

|  |  |
| --- | --- |
| **wait()** | **sleep()** |
| wait() is a method from Object class | sleep() is static method of Thread class |
| wait() method releases the lock when thread is waiting. | sleep() doesn’t release any lock while waiting. |
| wait method should be called from synchronized block or method else we will get IllegalMonitorStateException | There is no such requirement and sleep can be called from anywhere |
| To wake a thread from wait(), you have to call notify() or notifyAll() method | While in sleep (), thread gets start after the specified duration |

## 7. Difference between notify and notifyAll in Java?

notify method will wake up or notify only one thread and notifyall will notify all threads. If you are sure that more than one thread is waiting on monitor and you want all of them to give equal chance to compete for CPU, use notifyAll method. If multiple threads are waiting on a lock and if you call notify then it is not guaranteed that which thread will be informed.

## 8. Can we override private method in Java?

No, we cannot override private methods in Java. When we declare a method as private, it will be visible only for that class and not even to sub class. Hence overriding is not applicable in sub class.

## 9.What is the difference between using == operator and equals() method?

In case of primitive values, the == operator compares the values whereas in case of an object == compares the address of the objects and returns true if both references are pointing to the same object in the heap. equals() method on the other hand is used for logical comparison and its expected from an object to override this method to provide logical equality. For example, String class overrides the equals() method to return true if two Strings contains same letters.

## 10. The difference between Serializable and Externalizable interface in Java?

This is one of the frequently asked questions from Java Serialization.

Serializable is a marker interface(interface with no methods) is used to make Java classes serializable so that they can be transferred over network or their state can be saved on disk, but it uses default serialization built-in JVM. Externalizable interface contains two methods writeExternal() and readExternal() and allows you to fully control the Serialization process, specify a custom binary format and add more security measure and in terms of performance its good because everything is under control.

## 11. What do you mean by synchronization and what is the level of synchronization that can be applied?

Synchronization is nothing but control of access to a shared resource among multiple threads. Without synchronization when multiple thread try to access a shared resource there may arise unforeseen result due to concurrency issue. For example if multiple threads try to write within a same file then the file may be corrupted because one of the threads can overwrite data of other or while one thread is opening the file, at the same time another thread might be closing the file. So there is a need to control access to a shared resource such that only one thread can operate at a given time. Synchronization in java is achieved using *synchronized* keyword.

Synchronization can be applied to methods and blocks in a class.

Synchronized Method Example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | public class TestSynchronization {  public static void main(String[] args) {  Display d = new Display();  TestThread t1 = new TestThread(d, "Thread-1");  TestThread t2 = new TestThread(d, "Thread-2");  t1.start();  t2.start();  }  }  class Display {  synchronized void print(int n, String name) {// synchronized method  System.out.println("Running thread - " + name);  for (int i = 1; i <= n; i++) {  System.out.println(i);  try {  Thread.sleep(400);  } catch (Exception e) {  System.out.println(e); }  }  System.out.println("Exiting thread - " + name);  }  }  class TestThread extends Thread {  Display d;  TestThread(Display d, String name) {  this.d = d;  this.setName(name);  }  public void run() {  d.print(5, this.getName());  }} |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | Running the above program will give you the below output,  Running thread - Thread-1  1  2  3  4  5  Exiting thread - Thread-1  Running thread - Thread-2  1  2  3  4  5  Exiting thread - Thread-2 |

If you remove the synchronized keyword from the print method and run the program, the output will be different in different execution. Below is the output that is obtained in one of the executions.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | Running thread - Thread-1  Running thread - Thread-2  1  1  2  2  3  3  4  4  5  5  Exiting thread - Thread-1  Exiting thread - Thread-2 |

It is evident from the above output that, without synchronization, both the threads access the print() method simultaneously.

Synchronized Block Example:

The level of synchronization in the above example can also be achieved by using synchronized block. A synchronized block is nothing but a group of statements within curly braces with a synchronized keyword.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | public class TestSynchronization {  public static void main(String[] args) {  Display d = new Display();  TestThread t1 = new TestThread(d, "Thread-1");  TestThread t2 = new TestThread(d, "Thread-2");  t1.start();  t2.start();}  }  class Display {  void print(int n, String name) {  System.out.println("Running thread - " + name);  for (int i = 1; i <= n; i++) {  System.out.println(i);  try {  Thread.sleep(400);  } catch (Exception e) {  System.out.println(e);  } }  System.out.println("Exiting thread - " + name);  }}  class TestThread extends Thread {  Display d;  TestThread(Display d, String name) {    this.d = d;  this.setName(name);  }  public void run() {  synchronized (d) { // synchronized block  d.print(5, this.getName());    } }} |

When you run the above program you will get the same output as that of synchronized method example. Here synchronization is achieved by obtaining a lock on the display object d. So at a particular point of time only one thread can access methods of the object d. So when the program is run, Thread-1 gets a lock on object d and Thread-2 has to wait until Thread-1 finishes.

## Prime number program in Java

A number is said to be a prime number if it is greater than 1 and is divisible only by 1 and itself. For example the number 2 is divisible by 1 and 2 only. If you take 4 it is divisible by 1, 2 and 4. Hence 4 is not a prime number.

Write a program to check if the given number is prime or not (or) write a program to display Prime numbers from 1 to n (or) write a program to display n prime numbers are important Java interview questions. Let us see the program for these.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | public class PrimeNumberCheck {  public static void main(String[] args) {  checkIfPrimeNumber(1);  checkIfPrimeNumber(3);  checkIfPrimeNumber(11);  checkIfPrimeNumber(15);  }    private static void checkIfPrimeNumber(int n) {  boolean prime = true;  if (n > 1) {  for (int i = 2; i <= n / 2; i++) {  if (n % i == 0) {  prime = false;  break;  }  }  } else {  prime = false;  }  if (prime) {  System.out.println(n + " is a prime number");  } else {  System.out.println(n + " is not a prime number");  } }}  Output:  1 is not a prime number  3 is a prime number  11 is a prime number  15 is not a prime number |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | import java.util.Scanner;  public class PrimeNumberDisplay {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  System.out.println("Enter the value of n:");  int n = scanner.nextInt();  scanner.close();  if (n <= 1) {  System.out.println("There is no prime number in this range");  } else {  System.out.println("Prime numbers from 1 to " + n + " are: ");  }  for (int i = 2; i <= n; i++) {  boolean prime = true;  for (int j = 2; j <= i / 2; j++) {  if (i % j == 0) {  prime = false;  break;  }  }  if (prime) {  System.out.println(i);  } } }}  Output:  Enter the value of n:  30  Prime numbers from 1 to 30 are:  2  3  5  7  11  13  17  19  23  29 |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34 | import java.util.Scanner;  public class NPrimeNumberDisplay {  public static void main(String[] args) {  Scanner scanner = new Scanner(System.in);  System.out.println("Enter the value of n:");  int n = scanner.nextInt();  scanner.close();  if (n <= 1) {  System.out.println("There is no prime number in this range");  } else {  System.out.println("First " + n + " prime numbers are: ");  }  int count = 1;  for (int i = 2; count <= n; i++) {  boolean prime = true;  for (int j = 2; j <= i / 2; j++) {  if (i % j == 0) {  prime = false;  break;  }  }  if (prime) {  System.out.println(i);  count++;  } }}}  Output:  Enter the value of n:  5  First 5 prime numbers are:  2  3  5  7  11 |

### 