

What is the difference between Serialization and Marshaling?

Asked 15 years, 8 months ago Modified 2 years, 11 months ago

Viewed 173k times



731



I know that in terms of several distributed techniques (such as RPC), the term "Marshaling" is used but don't understand how it differs from Serialization. Aren't they both transforming objects into series of bits?



Related:

[What is Serialization?](#)

[What is Object Marshalling?](#)

serialization

terminology

marshalling

rpc

Share

Improve this question

Follow

edited Jun 20, 2020 at 9:12



Community Bot

1 • 1

asked Apr 20, 2009 at 23:26



Peter

48.9k • 46 • 135 • 184

**551**

Marshaling and serialization are *loosely* synonymous in the context of remote procedure call, but semantically different as a matter of intent.



In particular, marshaling is about getting parameters from here to there, while serialization is about copying structured data to or from a primitive form such as a byte stream. In this sense, serialization is one means to perform marshaling, usually implementing pass-by-value semantics.



It is also possible for an object to be marshaled by reference, in which case the data "on the wire" is simply location information for the original object. However, such an object may still be amenable to value serialization.

As @Bill mentions, there may be additional metadata such as code base location or even object implementation code.

Share Improve this answer

Follow

edited Jun 4, 2017 at 10:34



[user207421](#)

311k ● 44 ● 320 ● 488

answered Apr 20, 2009 at 23:42



[Jeffrey Hantin](#)

36.5k ● 7 ● 77 ● 96

4 Is there a word that means serialize and deserialize at the same time? Need a name for an interface with those

methods. – [raffian](#) Jul 20, 2014 at 3:53

- 1 @raffian, do you mean an interface implemented by the object that undergoes serialization and deserialization, or by the object responsible for managing the process? The key words I would suggest are "Serializable" and "Formatter" respectively; decorate with leading **I**, capitalization changes, and so forth as necessary. – [Jeffrey Hantin](#) Jul 24, 2014 at 2:29
-

@JeffreyHantin An object responsible for managing the process is what I meant; I'm using ISerializer now, but that's only half right :) – [raffian](#) Jul 24, 2014 at 3:59

- 27 @raffian in telecommunications, we call a component that serializes and deserializes a "SerDes" or "serdes", usually pronounced sir-dez or sir-deez depending on preference. I suppose it is similar to "modem" (i.e. "Modulator-Demodulator") in its construction. – [davidA](#) Jun 6, 2018 at 22:24
-

- 6 @naki it's industry-wide - if you look at high speed FPGA datasheets, they will mention SERDES functionality, although those are all fairly modern, going back to the 1990s. Google [NGrams](#) suggests it became more popular in the 1980s, although I did find an instance in an IBM datasheet from [1970](#) – [davidA](#) Jul 22, 2019 at 3:03 ✎
-



319

Both do one thing in common - that is *serializing* an Object. Serialization is used to transfer objects or to store them. But:



- **Serialization:** When you serialize an object, only the member data within that object is written to the byte stream; not the code that actually implements the object.



- **Marshalling:** Term Marshalling is used when we talk about **passing Object to remote objects(RMI)**. In Marshalling Object is serialized(member data is serialized) + Codebase is attached.

So Serialization is a part of Marshalling.

CodeBase is information that tells the receiver of Object where the implementation of this object can be found. Any program that thinks it might ever pass an object to another program that may not have seen it before must set the codebase, so that the receiver can know where to download the code from, if it doesn't have the code available locally. The receiver will, upon deserializing the object, fetch the codebase from it and load the code from that location.

Share Improve this answer

Follow

edited Jan 15, 2022 at 21:06



[banan3'14](#)

4,913 ● 6 ● 33 ● 66

answered Feb 2, 2013 at 2:50



[Nasir Ali](#)

3,191 ● 1 ● 13 ● 2

70 +1 for defining what *CodeBase* means in this context
– [Omar Salem](#) Dec 30, 2013 at 14:35

3 Marshaling without serialization does happen. See Swing's `invokeAndWait` and Forms's `Invoke`, which marshal a synchronous call to the UI thread without involving serialization. – [Jeffrey Hantin](#) May 5, 2015 at 20:19

- 2 "not the code that actually implements the object" : Does it mean the class methods ? or what does this mean. Can you please explain. – [Vishal Anand](#) Sep 7, 2016 at 8:44
- 3 What do you mean the implementation of this object ? Could you give a specific example of Serialization and Marshalling ? – [Cloud](#) Apr 30, 2018 at 11:31
- 1 Marshalling **without serialization** happens in some contexts, such as when a function call transfers control flow between threading models (for example, between a shared thread pool and a single-pinned-thread library) within a single process. That's why I say they're loosely synonymous **in the context of RPC**. – [Jeffrey Hantin](#) Sep 10, 2019 at 22:33



From the [Marshalling \(computer science\)](#) Wikipedia article:

123



The term "marshal" is considered to be synonymous with "serialize" in the Python standard library¹, but the terms are not synonymous in the Java-related RFC 2713:

To "marshal" an object means to record its state and codebase(s) in such a way that when the marshalled object is "unmarshalled", a copy of the original object is obtained, possibly by automatically loading the class definitions of the object. You can marshal any object that is serializable or remote. Marshalling is like serialization, except marshalling also records

codebases. Marshalling is different from serialization in that marshalling treats remote objects specially. (RFC 2713)

To "serialize" an object means to convert its state into a byte stream in such a way that the byte stream can be converted back into a copy of the object.

So, marshalling also saves the *codebase* of an object in the byte stream in addition to its state.

Share Improve this answer

edited Jun 20, 2020 at 9:12

Follow



Community Bot

1 ● 1

answered Apr 20, 2009 at 23:31



Bill the Lizard


405k ● 211 ● 572 ● 889

2 You mean an Object, if unserialized, can just have state, there won't be any codebase i.e. none of its function can be called, it is just a structured data type. And, if the same object is marshalled then it will have its codebase along with structure and once can call its functions? – [bjan](#) Mar 8, 2014 at 4:41

21 "Codebase" does not really means "Code". From "How Codebase Works" (goo.gl/VOM2Ym) Codebase is, quite simply, how programs that use RMI's semantics of remote class loading find new classes. When the sender of an object serializes that object for transmission to another JVM, it annotates the serialized stream of bytes with information called the codebase. This information tells the receiver

where the implementation of this object can be found. The actual information stored in the codebase annotation is a list of URLs from which the classfile for the needed object can be downloaded. – [Giuseppe Bertone](#) Jun 26, 2014 at 17:04

2 @Neurone That definition is specific to Jini and RMI. "Codebase" is a general term. en.wikipedia.org/wiki/Codebase – [Bill the Lizard](#) Jun 26, 2014 at 17:29

2 @BilltheLizard Yeah, but because you are talking about marshalling in Java, it's wrong to say that the difference between serialization and marshalling is "marshalling saves the code of the object in addition to its state", and it leads to the bjan's question. Marshalling saves the "codebase" in addition to the object state. – [Giuseppe Bertone](#) Jun 26, 2014 at 17:39 



Basics First

40



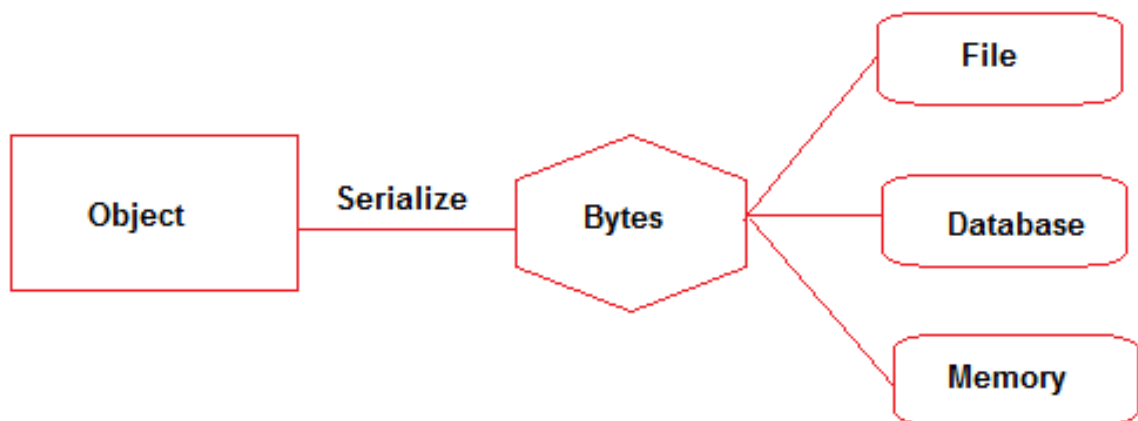
Byte Stream - Stream is a sequence of data. Input stream - reads data from source. Output stream - writes data to destination. Java Byte Streams are used to perform input/output byte by byte (8 bits at a time). A byte stream is suitable for processing raw data like binary files. Java Character Streams are used to perform input/output 2 bytes at a time, because Characters are stored using Unicode conventions in Java with 2 bytes for each character. Character stream is useful when we process (read/write) text files.

RMI (Remote Method Invocation) - an API that provides a mechanism to create distributed application in java. The

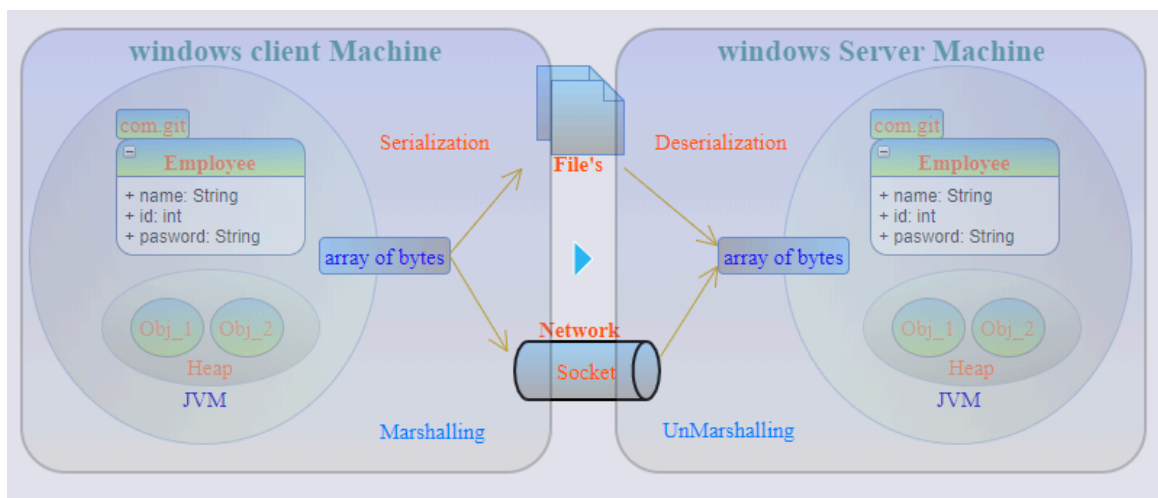
RMI allows an object to invoke methods on an object running in another JVM.

Both **Serialization** and **Marshalling** are loosely used as synonyms. Here are few differences.

Serialization - Data members of an object is written to binary form or Byte Stream (and then can be written in file/memory/database etc). No information about data-types can be retained once object data members are written to binary form.



Marshalling - Object is serialized (to byte stream in binary format) with data-type + Codebase attached and then passed **Remote Object (RMI)**. Marshalling will transform the data-type into a predetermined naming convention so that it can be reconstructed with respect to the initial data-type.



So Serialization is a part of Marshalling.

CodeBase is information that tells the receiver of Object where the implementation of this object can be found. Any program that thinks it might ever pass an object to another program that may not have seen it before must set the codebase, so that the receiver can know where to download the code from, if it doesn't have the code available locally. The receiver will, upon deserializing the object, fetch the codebase from it and load the code from that location. (Copied from @Nasir answer)

Serialization is almost like a stupid memory-dump of the memory used by the object(s), while **Marshalling** stores information about custom data-types.

In a way, Serialization performs marshalling with implementation of pass-by-value because no information of data-type is passed, just the primitive form is passed to byte stream.

Serialization may have some issues related to big-endian, small-endian if the stream is going from one OS to another if the different OS have different means of

representing the same data. On the other hand, marshalling is perfectly fine to migrate between OS because the result is a higher-level representation.

Share Improve this answer

Follow

edited Jan 15, 2022 at 21:16



banan3'14

4,913 ● 6 ● 33 ● 66

answered Jan 2, 2020 at 6:31



Om Sao

7,613 ● 2 ● 48 ● 68



21

Marshaling refers to converting the signature and parameters of a function into a single byte array.

Specifically for the purpose of RPC.



Serialization more often refers to converting an entire object / object tree into a byte array

*Marshaling will serialize object parameters in order to add them to the message and pass it across the network. *Serialization can also be used for storage to disk.**

Share Improve this answer

Follow

answered Oct 8, 2011 at 15:14



H.Gankanda

227 ● 2 ● 2



21

I think that the main difference is that Marshalling supposedly also involves the codebase. In other words, you would not be able to marshal and unmarshal an object into a state-equivalent instance of a different class.



Serialization just means that you can store the object and reobtain an equivalent state, even if it is an instance of another class.



That being said, they are typically synonyms.

Share Improve this answer

edited Jan 15, 2022 at 21:09

Follow



banan3'14

4,913 ● 6 ● 33 ● 66

answered Apr 20, 2009 at 23:30



Uri

89.6k ● 51 ● 226 ● 322

-
- 2 Do you mean an Object, if unserialized, can just have state, there won't be any codebase i.e. none of its function can be called, it is just a structured data type. And, if the same object is marshalled then it will have its codebase along with structure and one can call its functions? – [bjan](#) Mar 8, 2014 at 4:44
-

if the intent is just to share or save the states/semantics of an object this is serialization but if we also want to get the exact same type of object later we say we marshall. When 2 computer exchange IP packets, on computer A the IP packet can be represented with Struct ip_pkt when on the peer computer it is with Struct ip_data. ip_pkt and ip_data can layout their members in different ways but at the end both should have all the IP's parameters...here we have serialization! as we can't guaranteed that any two computers have same structs. in network we transfert meaning not object – [Narcisse Doudieu Siewe](#) Apr 28 at 21:04



16

Marshalling is the rule to tell compiler how the data will be represented on another environment/system; For example;



```
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 260)]
public string cFileName;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 14)]
public string cAlternateFileName;
```

as you can see two different string values represented as different value types.

Serialization will only convert object content, not representation (will stay same) and obey rules of serialization, (what to export or no). For example, private values will not be serialized, public values yes and object structure will stay same.

Share Improve this answer

answered Mar 12, 2015 at 17:54

Follow



Teoman shipahi

23k ● 16 ● 144 ● 172



9

Here's more specific examples of both:

Serialization Example:



```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
```



```
typedef struct {
    char value[11];
} SerializedInt32;

SerializedInt32 SerializeInt32(int32_t x)
{
    SerializedInt32 result;

    itoa(x, result.value, 10);

    return result;
}

int32_t DeserializeInt32(SerializedInt32 x)
{
    int32_t result;

    result = atoi(x.value);

    return result;
}

int main(int argc, char **argv)
{
    int x;
    SerializedInt32 data;
    int32_t result;

    x = -268435455;

    data = SerializeInt32(x);
    result = DeserializeInt32(data);

    printf("x = %s.\n", data.value);

    return result;
}
```

In serialization, data is flattened in a way that can be stored and unflattened later.

Marshalling Demo:

(MarshalDemoLib.cpp)

```
#include <iostream>
#include <string>

extern "C"
__declspec(dllexport)
void *StdCoutStdString(void *s)
{
    std::string *str = (std::string *)s;
    std::cout << *str;
}

extern "C"
__declspec(dllexport)
void *MarshalCStringToStdString(char *s)
{
    std::string *str(new std::string(s));

    std::cout << "string was successfully constructed.

    return str;
}

extern "C"
__declspec(dllexport)
void DestroyStdString(void *s)
{
    std::string *str((std::string *)s);
    delete str;

    std::cout << "string was successfully destroyed.\n
}
```

(MarshalDemo.c)

```
#include <Windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
```

```

int main(int argc, char **argv)
{
    void *myStdString;

    LoadLibrary("MarshalDemoLib");

    myStdString = ((void (*)(char *))GetProcAddress (
        GetModuleHandleA("MarshalDemoLib"),
        "MarshalCStringToStdString"
    ))("Hello, World!\n");

    ((void (*)(void *))GetProcAddress (
        GetModuleHandleA("MarshalDemoLib"),
        "StdCoutStdString"
    ))(myStdString);

    ((void (*)(void *))GetProcAddress (
        GetModuleHandleA("MarshalDemoLib"),
        "DestroyStdString"
    ))(myStdString);
}

```

In marshaling, data does not necessarily need to be flattened, but it needs to be transformed to another alternative representation. all casting is marshaling, but not all marshaling is casting.

Marshaling doesn't require dynamic allocation to be involved, it can also just be transformation between structs. For example, you might have a pair, but the function expects the pair's first and second elements to be other way around; you casting/memcpy one pair to another won't do the job because fst and snd will get flipped.

```

#include <stdio.h>

typedef struct {

```

```

    int fst;
    int snd;
} pair1;

typedef struct {
    int snd;
    int fst;
} pair2;

void pair2_dump(pair2 p)
{
    printf("%d %d\n", p.fst, p.snd);
}

pair2 marshal_pair1_to_pair2(pair1 p)
{
    pair2 result;
    result.fst = p.fst;
    result.snd = p.snd;
    return result;
}

pair1 given = {3, 7};

int main(int argc, char **argv)
{
    pair2_dump(marshal_pair1_to_pair2(given));

    return 0;
}

```

The concept of marshaling becomes especially important when you start dealing with tagged unions of many types. For example, you might find it difficult to get a JavaScript engine to print a "c string" for you, but you can ask it to print a wrapped c string for you. Or if you want to print a string from JavaScript runtime in a Lua or Python runtime. They are all strings, but often won't get along without marshaling.

An annoyance I had recently was that JScript arrays marshal to C# as "`__ComObject`", and has no documented way to play with this object. I can find the address of where it is, but I really don't know anything else about it, so the only way to really figure it out is to poke at it in any way possible and hopefully find useful information about it. So it becomes easier to create a new object with a friendlier interface like `Scripting.Dictionary`, copy the data from the JScript array object into it, and pass that object to C# instead of JScript's default array.

(`test.js`)

```
var x = new ActiveXObject('Dmitry.YetAnotherTestObject');
x.send([1, 2, 3, 4]);
```

(`YetAnotherTestObject.cs`)

```
using System;
using System.Runtime.InteropServices;

namespace Dmitry.YetAnotherTestObject
{
    [Guid("C612BD9B-74E0-4176-AAB8-C53EB24C2B29"), ComVisible]
    public class YetAnotherTestObject
    {
        public void send(object x)
        {
            System.Console.WriteLine(x.GetType().Name)
        }
    }
}
```

above prints "`__ComObject`", which is somewhat of a black box from the point of view of C#.

Another interesting concept is that you might have the understanding how to write code, and a computer that knows how to execute instructions, so as a programmer, you are effectively marshaling the concept of what you want the computer to do from your brain to the program image. If we had good enoughmarshallers, we could just think of what we want to do/change, and the program would change that way without typing on the keyboard. So, if you could have a way to store all the physical changes in your brain for the few seconds where you really want to write a semicolon, you could marshal that data into a signal to print a semicolon, but that's an extreme.

Share Improve this answer

edited Dec 29, 2021 at 23:27

Follow

answered Sep 10, 2017 at 6:41



Dmytro

5,213 ● 4 ● 44 ● 55



4



Marshalling is usually between relatively closely associated processes; serialization does not necessarily have that expectation. So when marshalling data between processes, for example, you may wish to merely send a REFERENCE to potentially expensive data to recover, whereas with serialization, you would wish to save it all, to properly recreate the object(s) when deserialized.

Share Improve this answer

answered Apr 20, 2009 at 23:32

Follow



Paul Sonier

39.5k ● 3 ● 79 ● 117



4



My understanding of marshalling is different to the other answers.

Serialization:

To Produce or rehydrate a wire-format version of an object graph utilizing a convention.

Marshalling:

To Produce or rehydrate a wire-format version of an object graph by utilizing a mapping file, so that the results can be customized. The tool may start by adhering to a convention, but the important difference is the ability to customize results.

Contract First Development:

Marshalling is important within the context of contract first development.

- Its possible to make changes to an internal object graph, while keeping the external interface stable over time. This way all of the service subscribers won't have to be modified for every trivial change.
- Its possible to map the results across different languages. For example from the property name convention of one language ('property_name') to another ('propertyName').

[Share](#) [Improve this answer](#)

[edited Jan 26, 2017 at 4:38](#)

[Follow](#)

answered Jun 25, 2014 at 0:14



[Jasper Blues](#)

28.7k ● 22 ● 109 ● 190

1 // , May I know more about what, specifically, "rehydrate" means, in this here answer, @JasperBlues? I'm guessing it's not just for Astronaut food. – [Nathan Basanese](#) Jan 26, 2017 at 0:59

1 @NathanBasanese according to this answer - stackoverflow.com/a/6991192/5101816 - definition of (re)hydrating contains in the following words: Hydrating an object is taking an object that exists in memory, that doesn't yet contain any domain data ("real" data), and then populating it with domain data (such as from a database, from the network, or from a file system). – [pawel-schmidt](#) Dec 3, 2018 at 8:27



3



Serialisation vs Marshalling

Problem: Object belongs to some process(VM) and it's lifetime is the same

Serialisation - transform **object state** into **stream of bytes**(JSON, XML...) for saving, sharing, transforming...

Marshalling - contains **Serialisation + codebase**.

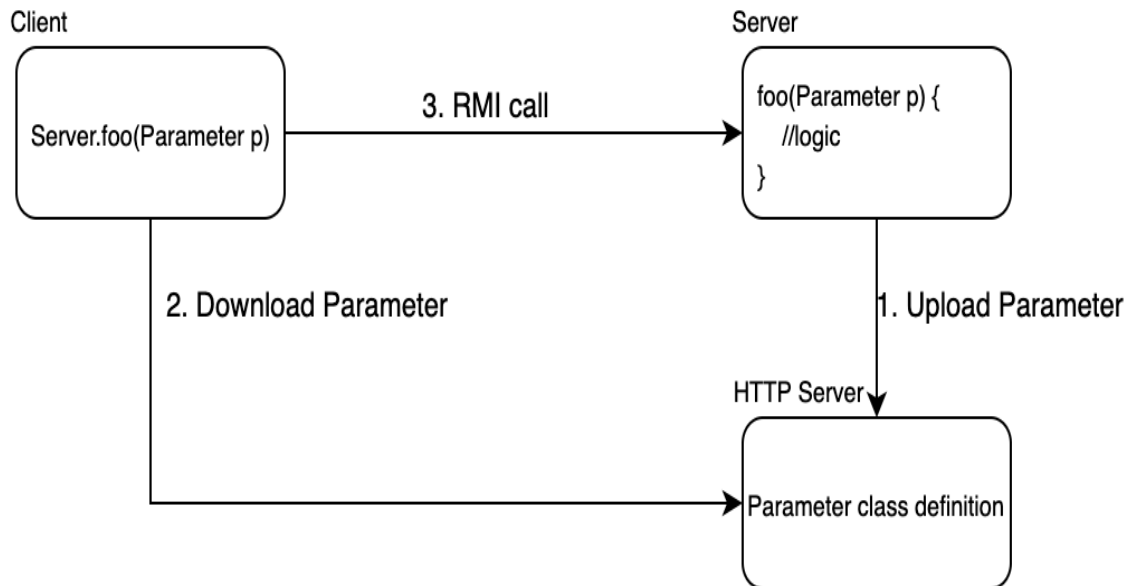
Usually it used by **Remote procedure call (RPC)** -> **Java Remote Method Invocation (Java RMI)** where you are able to invoke a object's method which is hosted on **remote Java processes**.

codebase - is a place or URL to **class definition** where it can be downloaded by **ClassLoader**. **CLASSPATH** [\[About\]](#) is as a **local codebase**

JVM -> Class Loader -> load class definition

```
java -Djava.rmi.server.codebase="<some_URL>" -jar  
<some.jar>
```

Very simple diagram for RMI



Serialisation - state
Marshalling - state + class definition

[Official doc](#)

Share Improve this answer

edited Mar 26, 2021 at 8:54

Follow

answered Jan 29, 2021 at 16:52



yoAlex5

34k ● 10 ● 223 ● 239



1



Marshaling uses Serialization process actually but the major difference is that in Serialization only data members and object itself get serialized not signatures but in Marshalling Object + code base(its implementation) will also get transformed into bytes.

Marshalling is the process to convert java object to xml objects using JAXB so that it can be used in web services.

Share Improve this answer

answered Nov 14, 2017 at 11:01

Follow



Aman Goel

3,531 ● 1 ● 23 ● 17



0



Think of them as synonyms, both have a producer that sends stuff over to a consumer... In the end fields of instances are written into a byte stream and the other end does the reverse and ends up with the same instances.

NB - java RMI also contains support for transporting classes that are missing from the recipient...

Share Improve this answer

answered Apr 20, 2009 at 23:46

Follow



mP.

18.2k ● 12 ● 77 ● 109