

其他地址

[面试问题手册](#)

[图灵学院的学习课程笔记](#)

[卢菲菲记忆学习](#)

前端

Jquery

Jquery的selector。

等\$(p) 选择p标签 “.”表示class, “#”表示id,

“:”表示行内里面的元素类型等\$(button)如：`sss` 另有\$(p:odd)、\$(p:even)、\$("ul li:first")、\$("ul li:first-child")

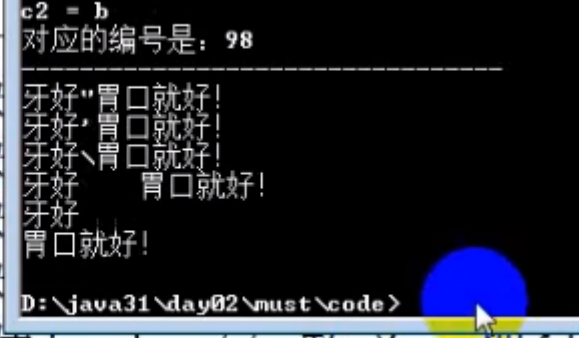
\$("a[target!='_blank']")：表示其中的target。

`HTML Tutorial`

JAVA 基础

转义

```
. println("-----")
. println("牙好\"胃口就好")
. println("牙好\'胃口就好")
. println("牙好\\胃口就好")
. println("牙好\t胃口就好")
. println("牙好\n胃口就好")
```



Java中需要转义的符号有：**单引号** (')、**双引号** (")、**反斜杠** (\)。在前面加一个反斜杠(\)就好了

其他像：“.”、“|”、“&”这种程序的操作符号在字符串中引用时是不需要转义的。

被正则表达式使用的时候要再基础上加2个反斜杠(\\),碰到需要转义的字符还要再加一个(也就是3个反斜杠)

```
19
20 @Override
21 protected Class<?> findClass(String name) throws ClassNotFoundException{
22 // String buildPath = dir + File.separator + name.replaceAll("\\.", Matcher.quoteReplacement(File.separator)) + ".class";
23 // String buildPath = dir + "\\\" + name.replaceAll("\\.", "\\\"") + ".class";
24 String buildPath = dir + "\\\" + name.replaceAll('.', '\\') + ".class";
25 File file = new File(buildPath);
26 try {
27     FileInputStream fi = new FileInputStream(file);
28     byte[] arr = new byte[fi.available()];
29     fi.read(arr,0,arr.length);
}
```

语法规范

if...else的缩写

1. 经典模式

```
if(Boolean){
    A;
}
else {
    B;
}
```

2. 精简模式

```
if(!Boolean){
    B;
}
A;
```

经典模式--->精简模式的简写，**需要保证 B 执行完就退出**，不往下走才能如此转换。

不然转换完至模式B时，A和B可能都会执行一遍（当（！Boolean）的情况）。

可以转换的场景：B模块 **return**、**Continue**、**break**

throw 异常

return:跳出方法、continue:跳出本次循环、break：跳出循环向下走。

原始类型(字母小写的8大基本类型：int、long、short、boolean、char、byte、float、double)**并不继承Object**

JAVA概念

2019-05-26

函数的方法签名method Signatrue：方法名、方法参数（）

In Java, a **method signature** is part of the **method** declaration. It's the combination of the **method** name and the parameter list. The reason for the emphasis on just the **method** name and parameter list is because of overloading.

Comparator接口:二个抽象方法，compare、equals。其中equals是继承自Object

函数式接口(Functional Interface)是Java 8对一类特殊类型的接口的称呼。这类接口只定义了唯一的抽象方法的接口（除了隐含的Object对象的公共方法），因此最开始也就做**SAM**类型的接口（Single Abstract Method）。

- 1.Comparator接口不用重载其中的equals方法，因为Object有equals。
 - 2.接口加载的时候，会自带Object下面的方法签名（method Signature），equals是一个有方法体的方法。
 - 3.Static Method：接口里面的静态方法不能被重写。
- Static Method有方法体，default Method也有方法体。

Static Method如：

```
public static <T extends Comparable<? super T>> Comparator<T> reverseOrder() {  
    return Collections.reverseOrder();  
}
```

****Default Method****如：

```
default Comparator<T> thenComparingInt(ToIntFunction<? super T> keyExtractor) {  
    return thenComparing(comparingInt(keyExtractor));  
}
```

Abstract Method必须重写！！且没有方法体。

```
int compare(T o1, T o2);
```

2019-05-27

JAVA转译符号：？

路径：路径中不区分大小写；"\"就是"\"，用双斜杠是避免转义符

static关键字：加上static关键字后，属性不能被序列化！

transient：Attributes and methods are skipped when serializing the object containing them。避免被序列化

Serialization is the process of converting an object into a stream of bytes 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.

私有的构造函数，不能实例化。

构造函数里面不能有static，构造函数是创建对象的，存到堆中。如果加上static关键字，则会存到方法区（静态区）中，矛盾。

```
public class Collections {  
    // Suppresses default constructor, ensuring non-instantiability.  
    private Collections() {  
    }  
}
```

不同于方法，构造器不能是abstract, static, final的。

- 1.构造器不是通过继承得到的，所以没有必要把它声明为final的。
- 2.同理，一个抽象的构造器将永远不会被实现，所以它也不能声明为abstract的。
- 3.构造器总是关联一个对象而被调用，所以把它声明为static是没有意义的。

2019-05-28

问题

- 1.为什么子类说抛出的异常不能比父类大。还有之类的权限能比父类大吗。

Java 方法重写注意事项

1. 子类中的方法与父类中的方法，有相同的返回类型、相同的方法名称、相同的参数列表
2. 子类中的方法的访问级别，不能低于父类中该方法的访问级别
3. 子类中方法抛出的异常范围，不能大于父类中方法抛出的异常的范围

注：子类中的方法的访问级别不能低于父类中该方法的访问级别，为何呢？假如没有这个限制，如果Child类的 outPut() 方法的访问级别为 private，将会与 java 的多态机制发生冲突。

Parent parent = new Child(); parent.outPut(); 这里Java 编译器会认为以上是合法的，但在运行时，根据动态绑定规则，Java虚拟机会调用 parent 变量所引用的 Child 实例的 outPut() 方法，而 Child 的 outPut() 方法为 private，Java虚拟机无法访问。为了避免这样的矛盾，Java 虚拟机不允许子类方法缩小父类中被覆盖方法的访问权限。

```
class test{
```

```
public static void main(String args[] ){
```

```
System.out.println("");  
}  
}
```

2019-06-03

1.ObjectInputStream和ObjectOutputStream的理解。

ObjectInputStream字面意思是**输入流**；ObjectOutputStream字面意思是**输出流**；

所谓的输入和输出是相对于Java自身来说，且以Java对象（内存）为参考。

文件流转为对象的过程，Java多出一个对象，此过程叫输入(Input)。反之，对象转文件流，Java并没有多出对象，但根据对象生成了流对象，此过程叫输出。以Java为参考即可。

Input和read；Output和write 绑定在一起。read和write是相对于外部（硬盘）来说的

read（读取）硬盘和write（写入）硬盘

2.try...catch和throws用法和区别。

我们有两种方式处理异常，一是throws抛出交给上级处理，二是try...catch做具体处理。

3.GBK占用2个字节，UTF-8占用3个字节。

2019-06-04

C擅长写java，c#，python，matlab，javascript的解释器或编译器，驱动程序，操作系统内核；
c++适合对效率要求较高的程序框架；c#适合写windows程序；java适合写高度抽象的复杂系统；
python适合做算法验证；matlab适合矩阵运算，数字信号处理等的建模，

接口

1. 返回接口的想上转型和直接返回。

见ReturnInterfaceImpl 本地Eclipse 工程InterfaceAndClass

2020-02-23

反射

线程

I/O

Collection

Junit

1. Junit (Java 单元测试) 中的Assert。黑盒、白盒测试, Junit属于白盒测试。
2. @Before和@After, 加入这二个注释可以在申请资源和释放资源时候写入, 无论测试的程序是否出现异常, 仍然会执行@After中声明的方法。

反射

1. 获取Class对象的方式：
 1. Class.forName("全类名"), 将字节码文件加载进内存并返回对象
 2. 类名.class, 从内存中调用已加载的字节码class对象
 3. 对象.getClass()
2. Class类是描述所有加载进内存的字节码文件 (.class) 的类。表示字节码文件对象。
硬盘<内存<cpu (Cache)
3. 同一个字节码文件 (*.class、Class对象) 在一次程序运行时, 只会加载一次。无论通过什么方式获取的Class对象, Class对象只有一个。
4. Class.getDeclaredField(String name), 无视访问权限修饰符得到其成员变量
Field.setAccessible(boolean flag), 暴力反射, 忽略权限进行修改。
5. Class.getConstructor。主要用来创建对象。
6. java程序在计算机经历的三个阶段: SOURCE (硬盘)、CLASS (内存)、RUNTIME

注解

1.
 - * 作用分类：
 - ①编写文档：通过代码里标识的注解生成文档【生成文档doc文档】
 - ②代码分析：通过代码里标识的注解对代码进行分析【使用反射】
 - ③编译检查：通过代码里标识的注解让编译器能够实现基本的编译检查【Override】
2. ANSI编码 American National Standards Institute 本地的编码格式。
3. javadoc XXX.java 生成API文档。“Shift+右键” 打开命令指示符。
4. JDK预定义的注解, @override, 检查是否继承关系。
@Deprecated, 表示不建议使用的, 如Date.month()。
@SuppressWarnings("all")压制所有警告。
5. javap XXX.class 将XXX字节码文件反编译成Java文件。Java parse
6. 注解 public @interface Myanno(){} == public interface Myanno extends Annotation
- 7.

属性：接口中的抽象方法

* 要求：

1. 属性的返回值类型有下列取值

- * 基本数据类型
- * String
- * 枚举
- * 注解
- * 以上类型的数组

2. 定义了属性，在使用时需要给属性赋值

1. 如果定义属性时，使用default关键字给属性默认初始化值，则使用注解时，可以不进行属性的赋值。
2. 如果只有一个属性需要赋值，并且属性的名称是value，则value可以省略，直接定义值即可。
3. 数组赋值时，值使用{}包裹。如果数组中只有一个值，则{}省略

8. * 元注解：用于描述注解的注解

* @Target：描述注解能够作用的位置

* ElementType取值：

- * TYPE：可以作用于类上
- * METHOD：可以作用于方法上
- * FIELD：可以作用于成员变量上

* @Retention：描述注解被保留的阶段

* @Retention(RetentionPolicy.RUNTIME)：当前被描述的注解，会保留到class字节码文件中，并被JVM读取到

* @Documented：描述注解是否被抽取到api文档中

* @Inherited：描述注解是否被子类继承

9.

```
//2. 获取上边的注解对象
//其实就是在内存中生成了一个该注解接口的子类实现对象
/*

    public class ProImpl implements Pro{
        public String className(){
            return "cn.itcast.annotation.Demo1";
        }
        public String methodName(){
            return "show";
        }
    }

    Pro an = reflectTestClass.getAnnotation(Pro.class);
```

10.

小结：

1. 以后大多数时候，我们会使用注解，而不是自定义注解
2. 注解给谁用？
 1. 编译器
 2. 给解析程序用
3. 注解不是程序的一部分，可以理解为注解就是一个标签

11. 使用BufferedReader的好处以及原因：

先缓存能够减少IO的读写次数，而IO操作是低效的，所以这样做可以提高效率。

BufferedReader会一次性从物理流中读取8k(默认数值,可以设置)字节内容到内存。

2019-07-04

XML

BLOB: Binary Large Object

CLOB: Character Large Object

XSD (XML Schema Definition)

&+; 中的amp就是英文**ampersand**的缩写,该词的意思是&这个符号

<+; :less than

>+; :great than

** +;** :Non-Breaking SPace的缩写 , 可以直接翻译成“不被折断的空格”

XML的二种约束手段 :

Document Type Definition (**DTD**)

Schema约束

2019-07-07

PCDATA (parsed character data) dtd里面的

Schema中的**elementFormDefault**

elementFormDefault取值 : qualified 或者 unqualified

All "qualified" elements and attributes are in the targetNamespace of the schema and all "unqualified" elements and attributes are in no namespace. All global elements and attributes are qualified.

a. 当**设置为unqualified时**, user为全局元素 (可作为根元素) 必须添加前缀, 非全局元素

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.org/01"
  xmlns:tns="http://www.example.org/01"
  elementFormDefault="unqualified">

  <element name="user">
    <complexType>
      <sequence>
        <element name="id" type="int"></element>
        <element name="name" type="string"></element>
      </sequence>
    </complexType>
  </element>

</schema>
```

<http://blog.csdn.net/lmj623565791>


```
<?xml version="1.0" encoding="UTF-8"?>
<ns:user xmlns:ns="http://www.example.org/01"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.org/01 01.xsd">
  <id>1</id>
  <name>1</name>
</ns:user>
```

<http://blog.csdn.net/lmj623565791>

b. 当设置为qualified时，所有的元素都必须添加前缀。

```
<?xml version="1.0" encoding="UTF-8"?>
<ns:user xmlns:ns="http://www.example.org/01"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.example.org/01 01.xsd">
  <ns:id>1</ns:id>
  <ns:name>1</ns:name>
</ns:user>
```

<http://blog.csdn.net/lmj623565791>

Use `elementFormDefault` to act as a switch for controlling namespace exposure - if you want element namespaces exposed in instance documents, simply turn the `elementFormDefault` switch to "on" (i.e., `setElementFormDefault="qualified"`); if you don't want element namespaces exposed in instance documents, simply turn the `elementFormDefault` switch to "off" (i.e., `setElementFormDefault="unqualified"`).

各类网络协议以及关系

TCP/IP、UDP、http、https、webservice、soap、socket、socks

·什么是 webservice?

- Web service 即 web 服务，因为互联网而产生，发布 web 服务后可以将资源进行共享，通过 webservice 调用获取并操作资源信息。
- webservice 是一种跨编程语言和跨操作系统平台的远程调用技术即跨平台远程调用技术。
- 采用标准 SOAP(Simple Object Access Protocol) 协议传输，soap 属 w3c 标准。基于 http 传输 xml，即 soap=http+xml。
- 采用 wsdl 作为描述语言即 webservice 使用说明书，wsdl 属 w3c 标准。
- xml 和 XSD(XML Schema Datatypes)是 webservice 的跨平台的基础，XML 主要的优点在于它既与平台无关，又与厂商无关。XML 是由万维网协会(W3C)创

WebService是对socket的一个上层协议，上层的封装-----

http 协议基于socket，此外，web service基于http协议和soap。

Socket是基于TCP/IP的传输层协议。

WebService是基于HTTP协议传输数据的，HTTP是基于TCP的应用层协议。

SOAP（协议）：查看报文的时候知道soap就是一种体。用xml格式传输数据报文。

socket:

A socket is **one endpoint** of a two-way communication link between two programs running on the network.

[socket]: Socket - <https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html> Socket

表现形式如：192.168.0.1:8080 这个就是一个socket端点地址。

每次双向连接会产生一个相同的socket，地址还是192.168.0.1:8080

socket和端口是有点相似的。区别：The **main difference** between socket and port is that the **socket is the interface of sending and receiving data on a specific port while the port is a numerical value assigned to a specific process or an application in the device.**

In brief, a socket is the **communication path** to a port.

TCP/IP protocol: <https://searchnetworking.techtarget.com/definition/TCP>

TCP和UDP就好比2种不同的交通工具。如传输一张图片，TCP传输时，图片完整。而UDP则有可能将图片变得面目全非（比喻）。一个面向连接，可以判断双方数据是否传输完成。UDP则无法判断传输是否完成。

协议：协议可以看作是一种双方约定好的数据结构。

TCP/IP、UDP、http、soap、socks。都是协议，可以看作是一种数据结构。

读TCP/IP有感

PDU（Protocol Data Unit）协议数据单元：

The PDU that IP sends to link-layer protocols is called an IP datagram ---vol1 -53页

说明了在TCP/IP协议中。PDU=IP datagram 俗称packet(包)。多个包传到链路层的时候就称为frames（帧）。~~以上可以判断出，包就是TCP/IP协议中传输的基本单元。应该传到链路层的frames才是最基本的。因为每层往下走都会封装一次信息。~~

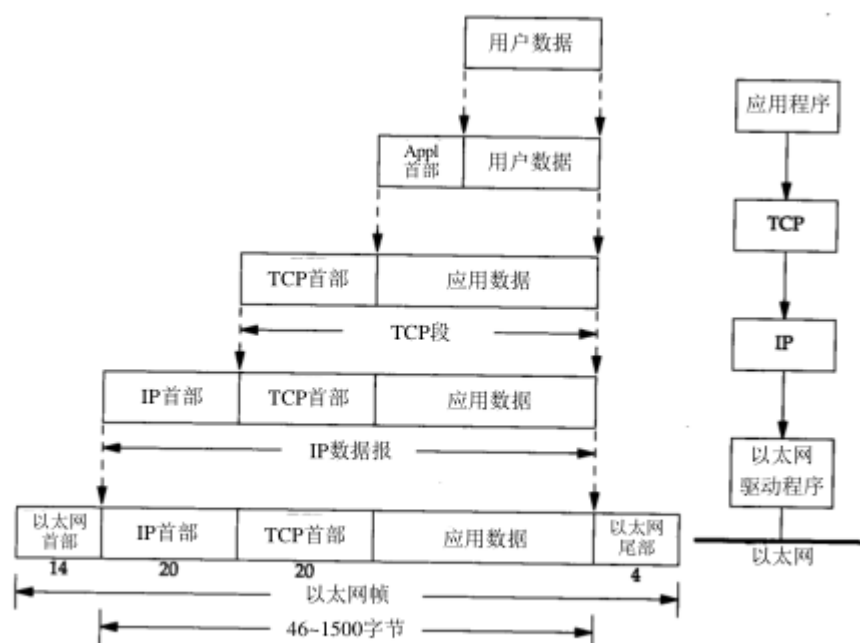


图1-7 数据进入协议栈时的封装过程

Demultiplexing: 拆帧

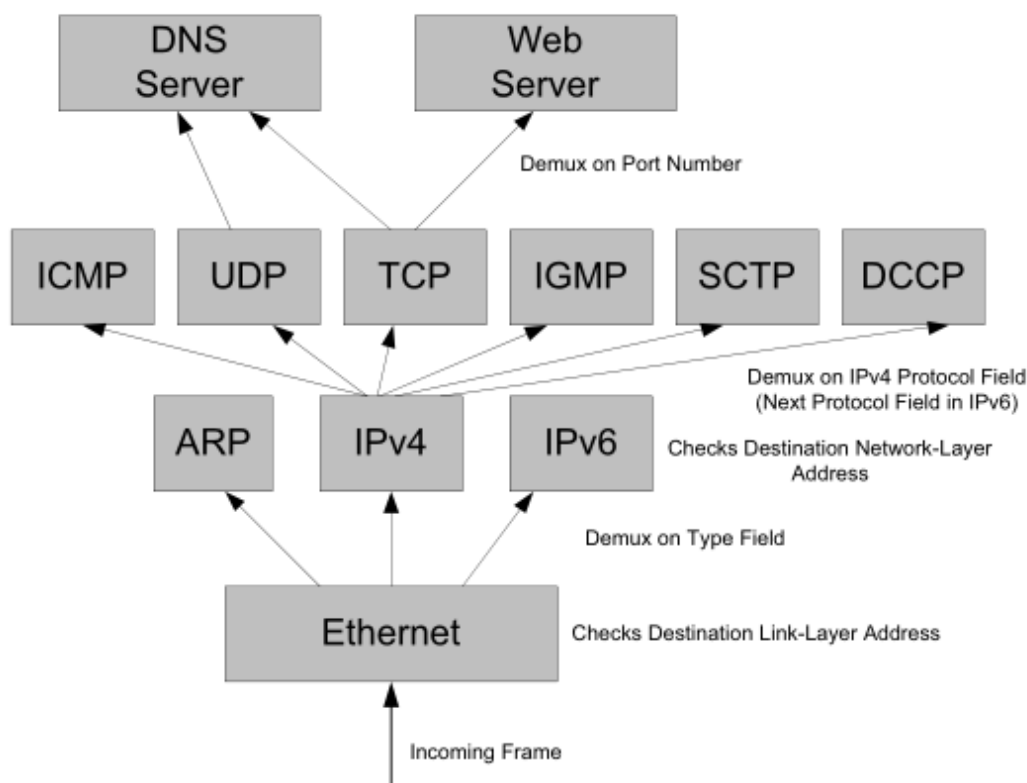


Figure 1-6 The TCP/IP stack uses a combination of addressing information and protocol demultiplexing identifiers to determine if a datagram has been received correctly and, if so, what entity should process it. Several layers also check numeric values (e.g., checksums) to ensure that the contents have not been damaged in transit.

TCP数据被封装在一个IP数据报中，如图17-1所示。

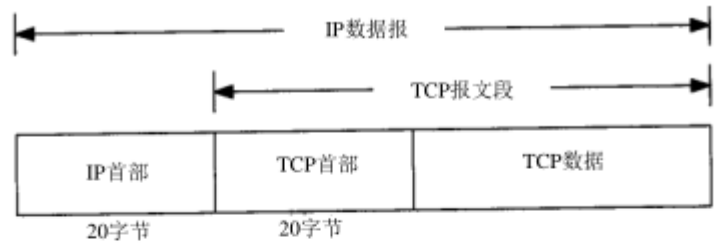


图17-1 TCP数据在IP数据报中的封装

图17-2显示TCP首部的数据格式。如果不计任选字段，它通常是 20 个字节。

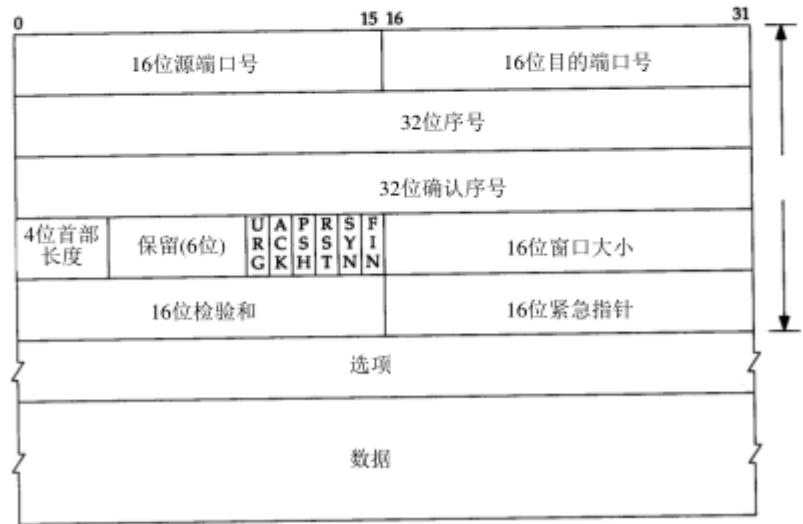


图17-2 TCP包首部

UDP首部的各字段如图11-2所示。



图11-2 UDP首部

有时，一个IP地址和一个端口号也称为一个插口（socket）。这个术语出现在最早的TCP规范（RFC793）中，后来它也作为表示伯克利版的编程接口（参见1.15节）。插口对（socket pair）（包含客户IP地址、客户端口号、服务器IP地址和服务器端口号的四元组）可唯一确定互连网络中每个TCP连接的双方。

框架

Spring

IOC

1. 学习中需要掌握的前置知识：

1. 设计模式

装饰者模式 (wrapper)

工厂和抽象工厂

观察者模式 () 对应IOC的listener

2. 反射、Jdk 的动态代理

反射的 Field.getType()

Filed.set(obj,obj) 这2个方法需要好好掌握。

jdk实现的动态代理是如何实现的。

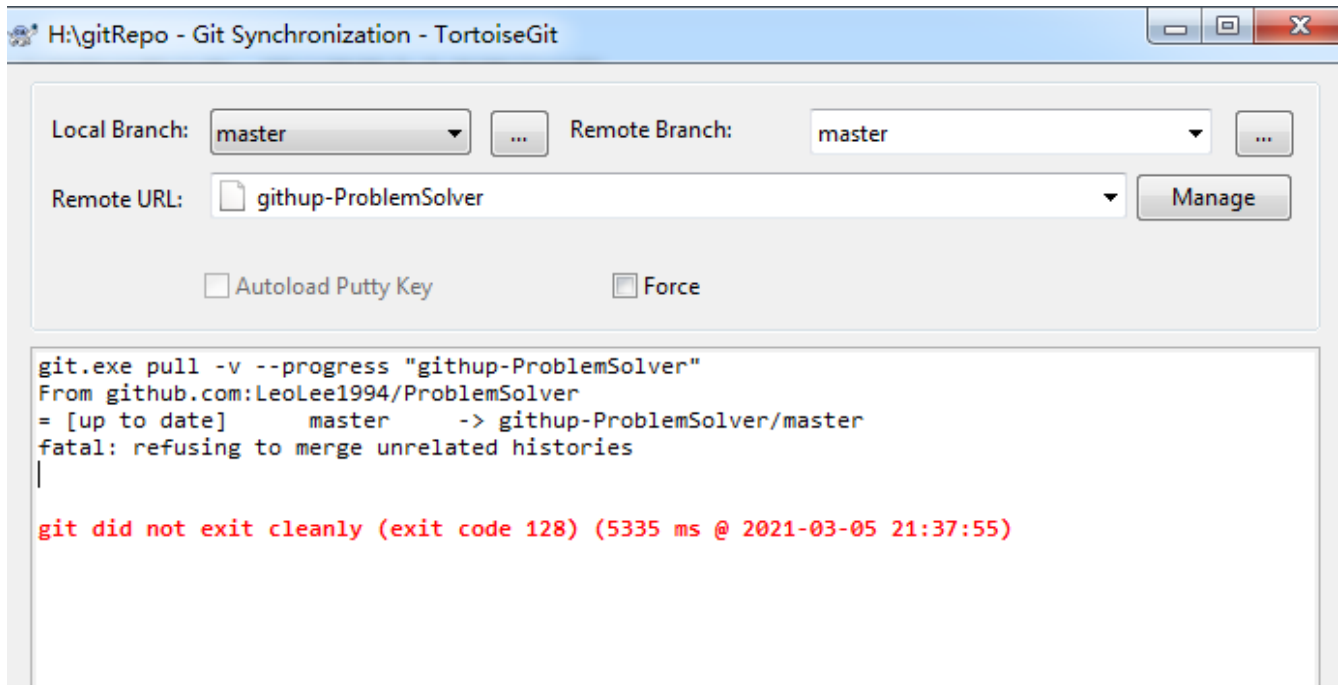
工具

git

[训练地址](#)

1. 如果一个没有关联的2个仓库相互拉取分支，可以使用命令来强行拉取

```
git pull githup-ProblemSolver master --allow-unrelated-histories
```



拉取远程代码报的错误：refusing to merge unrelated histories。

2. 没有克隆的右键-----使用命令：git clone --help

案例中的 git clone ssh-url [路径 (可不加)]

3. 删除一个仓库

删除仓库文件夹下隐藏的.git文件夹即可，.git文件夹删除了对应的这个仓库也就删除了。

4. 分支和冲突

分支，一个仓库（目录）下的多个展示。

注意：**合并代码的时候要先切换到master主干再merge其他分支**，merge完其他分支后可以删除其他分支。

冲突：产生的原因是因为再**同一个版本**下对同文件的同一行进行了修改。（同一版本不同行可以合并，类似cc的常量类不用解决版本冲突。但是同版本同文件不同行就需要区分版本的提交顺序）。解决办法：每次提交代码的时候最好先更新（push下来）。**不同行随便合并，相同行需要更新重提解决冲突**

5. git查看分支

1. 当前分支: git branch -vv
2. 所有分支: git branch
3. 查看远程分支: git branch -r
4. 查看所有分支: git branch -a

6. 基本操作

1. 创建本地分支: git branch [dev]
2. 切换分支: git checkout [dev]
3. 提交文件到分支: git add [filename] + git commit -m "" [filename]
4. 提交分支到远程(前面是用右键图形工具): git push github-ProblemSolver dev:master
5. 拉取远程分支：git pull github-ProblemSolver master

```
MINGW64:/h/gitRepo
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

Administrator@TAZWF600VTJPEVF MINGW64 /h/gitRepo (dev)
$ git push github-ProblemSolver dev:master
To github.com:LeoLee1994/ProblemSolver.git
! [rejected]        dev -> master (fetch first)
error: failed to push some refs to 'github.com:LeoLee1994/ProblemSolver.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

Administrator@TAZWF600VTJPEVF MINGW64 /h/gitRepo (dev)
$ git push github-ProblemSolver dev:master
To github.com:LeoLee1994/ProblemSolver.git
! [rejected]        dev -> master (fetch first)
error: failed to push some refs to 'github.com:LeoLee1994/ProblemSolver.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

Administrator@TAZWF600VTJPEVF MINGW64 /h/gitRepo (dev)
$ git pull github-ProblemSolver master
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 626 bytes | 156.00 KiB/s, done.
From github.com:LeoLee1994/ProblemSolver
* branch                master      -> FETCH_HEAD
4039c2c..2ac6f2c master    -> github-ProblemSolver/master
Already up to date!
Merge made by the 'recursive' strategy.

Administrator@TAZWF600VTJPEVF MINGW64 /h/gitRepo (dev)
$ git push github-ProblemSolver dev:master
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 4 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 418 bytes | 418.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To github.com:LeoLee1994/ProblemSolver.git
2ac6f2c..b1eac2d dev -> master
```

7. Git补充

1. git pull = git fetch + git merge
2. checkout，会在本地创建对应的远程分支

该命令会将当前工作分支切换到branchName。另外，可以通过下面的命令在**新分支创建的同时切换分支**：

```
git checkout -b newBranch
```

该命令相当于下面这两条命令的执行结果：

1. `git branch newBranch`
2. `git checkout newBranch`

3. pull和clone的区别是，clone会把远程整个.git复制过来，而pull则不会，而是从当前版本开始。

将当前分支(dev)的推送到远程分支(github-ProblemSolver.master)

```
git push github-ProblemSolver(远程仓库名) dev(当前分支名):master(远程仓库分支名)
```

4. git本地的master分支删除了一个文件但还没有提交，用git status可以查看修改的文件，并用指令 `git restore [filename]`恢复删除的文件。删除文件的命令：`git rm [filename]`；删除后提交: `git commit -m "(提交的备注)"`

5. git的版本回退，一次commit有对应一个commitid。使用命令:

```
git reset --hard head~1('1'表示前一个版本,可以是其他数字)
```

docker

官方文档 <https://docs.docker.com/engine/install/centos/>

1. Centos安装docker [教程](#)

2. [docker命令](#)

`run` ,启动一个image并根据命令生成一个容器实例，有对应容器id。

`start` ,运行前面创建的容器(镜像实例)。

`-it` ,交互式运行。比如镜像是linux,运行后可以进入容器就像进入另一个系统。

3. 用阿里云的docker快速创建mysql并开放端口实现快速访问；地址

学前的几个问题

1. docker为什么比vm快，除了架构上面的。他里面有直接使用的软件吗？

有其他软件image可以方便下载

2. 同享网络ip、端口吗？不共享的话可以设置分开的端口或者ip吗？

可以设置有不同ip，端口号。但是ip要再同一个网段上，并且网卡需要设置。

23种设计模式

阿里巴巴规范中所写的设计模式的作用，其本质就是封装变化。学习设计模式，每种模式分别多举出几个例子来进行比较。要求把每个例子的**目的（需求）、变化的点**进行梳理。以体会设计模式是如何处理这些变化的。梳理完后，再进行梳理和总结。 <20201015> 23种设计模式出自于4人帮（Gang of Four）。书名的**Reusable**和**Object-Oriented**值得思考。这本书是以别人的代码为材料，进行总结。所以现在学习的方法稍微做点调整，原来是根据视频中的例子，总结例子的需求、模仿实现、再总结和模拟。现在，我觉得应该根据JDK里面原有的设计模式进行总

结，就像当年这本书做的这样，循着这本书的做法，去体会设计模式。然后再去应用。看JDK的源码体会设计模式是由场景的，就比如策略模式里面的Arrays.sort(),他设计的场景就是为了我们开发者，所以更加能体会到作者的良苦用心，进而理解设计模式。进而应用。</20201015>

	Design Patterns: Elements of Reusable Object-Oriented Software
Author	The "Gang of Four": Erich Gamma Richard Helm Ralph Johnson John Vlissides
Country	United States

单例 (singleton)

详见，eclipse下23designsOfModel工程的package-info

策略模式 (strategy)

1. 马士兵

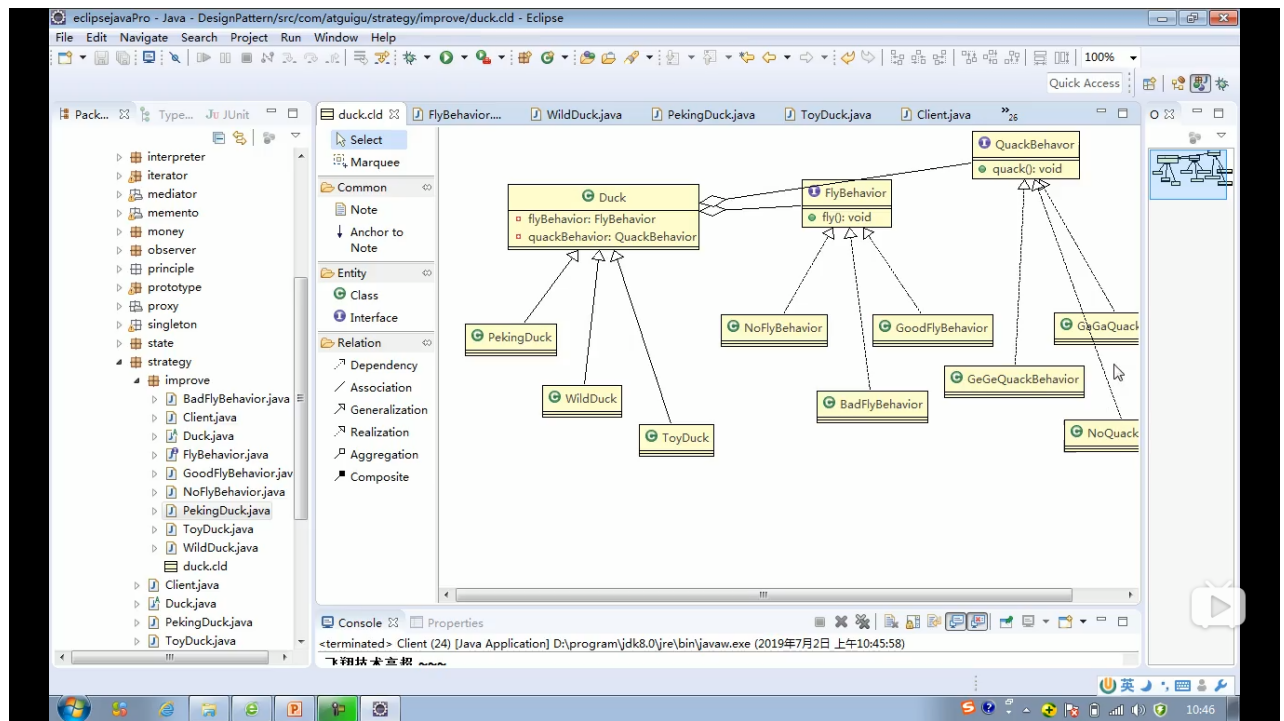
目的 (需求) :

1. 写一个sort类。对不同的对象**进行**排序。不同的对象排序会对应不同的排序办法。
2. 对不同的对象**进行**多种功能的排序。

变化的点：不得而知

能获得的已知：排序的对象，具体排序的方法

2. 尚硅谷



目的（需求）：不同的鸭子有不同的行为，具体不同的行为中又有不同的属性。

变化：不得而知

能获得的已知：不同的鸭子，不同的行为。

3. JDK Arrays类中的comparator().

工厂模式(Factory)

抽象工厂模式 (abstract Factory)

门面模式 (facade)

对外的接口

调停模式 (mediate)

对内的接口

装饰模式 (Decorator)

数据库知识复习

2019-06-13

常见的知识点

1. 左右连, inner/left/right/full join...on。

比较过程: 1.两个表格并排显示、2.确定主表, 主表全部显示、3.根据条件, 以主表条件为基准列出。

2. union, 以左边的字段为基准。行装列。

存储过程

Nvl(A,B)意思是如果A为Null, 那么返回B, 否则返回A的值。Null Of Value

“||”连接符号 可以将两个字段的值连接在一起 例: select 姓||名 from user;

: = 赋值 相当于 “= ”。 << 标签分隔符(开始) | | 标签分隔符(结束)

Fuction和Procedure:存储过程和存储函数

存储过程和函数的区别:

1. 它们本质上没有区别
2. 函数存在的意义是给过程调用 存储过程里面调用存储函数
3. 函数可以在sql语句里面直接调用
4. 存储过程能实现的, 存储函数也能实现, 存储函数能实现的, 过程也能实现

Oracle数组:

```
declare
type v_table is table of varchar2(30) index by binary_integer;
m_array v_table;
begin
    select tage bulk collect into m_array from TEST1;
    for i in 1..m_array.count
    loop
        /*m_array(i):=i; */
        dbms_output.put_line(m_array(i));
    end loop;
end;
```

存储过程和函数的格式。 create or replace procedure (para in|out **type**,...,)

as/is define

begin ... commit; end;

create or replace procedure (para in|out type,...,)

return **type**;

as/is define

begin ... return para; end;

```

create or replace function fsrcall(pempno in number,total in number)
return number
as
    rep number;
begin
    select sal into rep from emp where empno=pempno;
    return rep;
end;

create or replace procedure srcall(pempno in number,total out number)
as
    rep number;
begin
    select sal into total from emp where empno=pempno;
    commit;
end;

```

Mysql

1. 安装mysql [对应的安装记录](#)

2. mysql使用

1. 启动服务及连接数据库

```
net start mysql
```

```
mysql -u root -pshow tables
```

show databases;查看当前用户下的数据库

use "database", 使用该数据库

show tables ; 查看当前库下有哪些表

2. 事务开启的命令

1.begin或者start transaction : 显式开启一个事务

2.commit : 提交一个事务, 使修改是永久的

3.rollback : 回滚事务, 撤销未提交的修改

3. [隔离级别](#)

Redis

1. Redis高并发分布式锁

1. synchronize加锁方式无法满足多台机器(redis数据)一致性的问题。减库存操作, 会存在读同一值导致**超卖**的问题。

2. Jedis代码解决超卖问题思路:

1. 设一个键值作为分布式锁。stringRedisTemplate.opsForValue().SetIfAbsent(...)来判断是否加上了锁

2. 分布式锁锁失效时间设置问题

1. 太大---其实实际应该大点，程序中断会使得key不失效，后面线程一直拿不到锁。

2. 太小

当前线程没有执行完锁就失效了，后面的线程会拿到锁执行，当前线程执行完后，又会失效锁，导致后面的线程又会进来，相当于此锁失效了。

3. 失效问题解决。在finally中判断当前锁中value(uuid,线程id都可以)是否还是开始的那个value。只有是自己加的锁才去删除。

3. Redisson解决超卖问题

Redisson有一个重入锁可以实现。Redisson.getLock()得到该锁rlock，然后执行程序前上锁rlock.lock()---或者trylock()，finally解锁。里面的原理是用LUA脚本保证一个判断、取值、设值几步是一个原子操作；会有循环的定时任务判断锁的失效时间，如果在程序执行的时候失效了就会重写延长时间，以保持其他线程不会拿到该锁。

shell

2019-11-14

1. Sed的学习。

sed [option] "parttern command" file

将文件中的parttern替换：sed -i 's/原/replacement/g' filename

2. 尽量使用双引号""，不使用单引号''。

单引号中存在变量时，在单引号中不能转换其变量。在双引号中是可以转化其变量的。

bat-winshell-cmd

```
@echo OFF
set starPath=D:\002\

for %%i in (%starPath%*.*) do (
    Start "" "%%i"
    echo %%i >> %starPath%abcdf.txt
)
```

☐: <https://juejin.im/post/5afefdb46fb9a07aac24b647> "一键多开"

dir = l

rmdir = rm

help = man

数据结构

算法

网络

OSI七层协议模型

TCP/IP

第7层 应用层

各种应用程序协议，如 HTTP、FTP、SMTP、POP3。



7

第6层 表示层

信息的语法语义以及它们的关联，如加密解密、转换翻译、压缩解压缩。

6

第5层 会话层

不同机器上的用户之间建立及管理会话。

5

第4层 传输层

接受上一层的数据，在必要的时候把数据进行分割，并将这些数据交给网络层，且保证这些数据段有效到达对端。

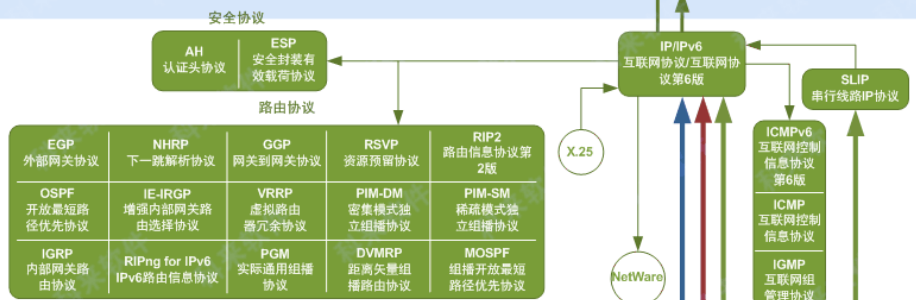
4

TCP 传输控制协议
UDP 用户数据报协议

第3层 网络层

控制子网的运行，如逻辑编址、分组传输、路由选择。

3



第2层 数据链路层

物理寻址，同时将原始比特流转变为逻辑传输线路。

2



第1层 物理层

机械、电子、定时接口通信信道上的原始比特流传输。

1

IEEE 802.2
Ethernet v.2
Internetwork

问题及方向记录

学习Java JDK的指令如：javap、javadoc....、

2020-02-11

了解复习注解的原理。学习NIO，复习老IO。线程池、锁。为学习springBoot打下基础。

http究竟是一个什么东西，和socket有什么联系？TCP协议又究竟干了什么。协议的本质是什么

2020-02-24

每天学习的东西要几下，并复习之前的知识。（用写代码的形式复习）

1. 重新学习了if/else的简写

1. 可以转成倒转体的转成倒装体。如前面总结的，先判断return、break

2. **Reference = Boolean ? Value1** (Boolean = true) : **Value2** (Boolean = false)

2. 线程池

1. 由executors.new...threadPool产生一个 什么 threadService。

本质是实例化 threadPoolExecutor(...)，参数里面很多概念，决定着线程池的性能。

2021-02-21

1. 安装mysql

[对应的安装记录](#)