Table of Contents

PanRufeng's Blog 1.1

Architect 1.2

- Autosys HTTPS - java.security.cert.CertificateException 1.2.1
- CSRF跨站请求伪造 1.2.2
- Excel - VBA导入CSV为文本格式及中文乱码问题 1.2.3
- HTTPS证书准备过程 1.2.4
- JMeter测试动态参数API 1.2.5
- JWT技术简单理解和实现 1.2.6
- LVS+Keepalived+Nginx高可用负载均衡方案 1.2.7
- Nginx - Basic Authentication为Kibana增加登录验证功能 1.2.8
- Nginx - cannot load certificate due to SELinux 1.2.9
- Nginx - LDAP Authentication 1.2.10
- Nginx - the ssl parameter requires ngx_http_ssl_module 1.2.11
- Nginx - 使用OpenSSL自签名证书配置HTTPS 1.2.12
- Send Mail from PowerShell Using SMTP 1.2.13
- Tomcat AJP连接器配置secretRequired=true,但是属性secret确实空或者空字符串 1.2.14
- Tomcat无法将位于[XX]的资源添加到Web应用程序[XX]的缓存中 1.2.15
- Who is Solution Architect 1.2.16
- 使用curl测试RESTful API 1.2.17
- 使用Jmeter对接口进行压力测试 1.2.18
- 重构 - Java程序太多if else怎么办 1.2.19

DevOps 1.3

- Anaconda多版本Python切换 1.3.1
- Git Command Alias 1.3.2
- Git error - RPC failed 1.3.3
- Git常用命令 1.3.4
- Python基础 1.3.5
- SSL证书PEM文件里有什么和要有什么 1.3.6
- Upgrade Python2.7.5 to Python2.7.16 1.3.7
- VMWare虚拟机CentOS联网：主机能SSH连虚拟机，虚拟机能上网 1.3.8

Win10不用虚拟机也可以装Linux Ubuntu 1.3.9
ELK 1.4
Define Mutilple Pipelines For Logstash 1.4.1
ElastAlert最大的坑 1.4.2
Elasticsearch - Validation Failed - cluster currently has maximum shards open 1.4.3
Elasticsearch JVM OutofMemory 1.4.4
elasticsearch.service start operation timed out 1.4.5
Elasticsearch什么时候使用Data Transform和Rollup Jobs 1.4.6
Elasticsearch启动不了 1.4.7
Elasticsearch权威指南里的简单搜索例子 1.4.8
Elasticsearch权威指南里的聚合分析报错 1.4.9
Filebeat重启之后不读Log了？ Registry file is too large 1.4.10
Hearthbeat实现服务监控 1.4.11
Logstash-keystore保存Elastisearch密码 1.4.12
Logstash收集多种Filebeat数据 1.4.13
Logstash配置调试技巧 1.4.14
Set up basic security for the Elastic Stack plus secured HTTPS traffic 1.4.15
Set up basic security for the Elastic Stack 1.4.16
Watcher发送告警邮件和调用接口 1.4.17
X-Pack设置用户密码 1.4.18
使用ElastAlert发送邮件 1.4.19
快速搭建ELK7.2 1.4.20
快速搭建ELK7.3 1.4.21
Java 1.5
Apache Log4j2 Remote Code Execution RCE Vulnerability - CVE-2021-44228 - ESA-2021-31 1.5.1
Logback SMTPAppender Send Mail When Log Error 1.5.2
Spring - Schedule Task 如何动态改写Cron定义 1.5.3
ThreadPoolExecutor的行为模式 1.5.4
定制JDK删除JRE里不必要的文件 Build Custom JDK JRE 1.5.5
JavaScript 1.6
Angular Helloworld 1.6.1
Angular Karma - Chrome have not captured in 60000 ms 1.6.2
Angular Tutorial里的测试坑 1.6.3

Angular只跑某一个Test Case 1.6.4
GitBook运行报错 - no such file or directory 1.6.5
JavaScript if变量 到底什么时候成立 1.6.6
JavaScript设计模式 - Strategy 1.6.7
JavaScript进阶 - Promise 1.6.8
Node.js Bunyan输出JSON日志 1.6.9
Node.js Log4js使用详解 1.6.10
Node.js Log4js自己写Appender 1.6.11
Node.js Log4js输出JSON日志 1.6.12
Node.js Log4js输出自定义格式和变量 1.6.13
Node.js RESTful API and Express Unit Test 1.6.14
Node.js Sinon测试替身 1.6.15
npm run传递参数 1.6.16
TypeScript Unit Test with Mocha, Istanbul and Chai 1.6.17
VSCode跳出右括号 1.6.18
VSCode集成Git Bash作为Terminal 1.6.19
使用Mocha和Istanbul实现Node.js单元测试和覆盖率 1.6.20
怎样安装NPM离线包 1.6.21
整合Bamboo和SonarQube实现Node.js自动化测试 1.6.22
Others 1.7
base_concerpt 1.7.1
保守投资策略研究 1.7.2
商业银行主要监管指标 1.7.3
指定Python安装包下载地址 1.7.4
最受欢迎数据库排行榜 1.7.5
读《麦肯锡笔记思考法》 1.7.6
Pending 1.8
TODO 1.8.1
Reading 1.9
2018年书单 1.9.1
2019年书单 1.9.2
2020年书单 1.9.3
不要和我比懒，我懒得和你比 1.9.4
不要老骂孩子，打就行了 1.9.5
不要老骂孩子，打就行了（二） 1.9.6

世俗的成功法则 1.9.7
人生中最重要的事，是洗碗 1.9.8
今天，我读了一首诗 1.9.9
住在广州塔下的同事，你还好吗 1.9.10
公司理财师专业能力 1.9.11
关于如何佛系地辅导作业 1.9.12
妹妹画画：喜欢即是欢喜 1.9.13
想不到古人已经在用护手霜了，还是专业护手霜 1.9.14
我们应该知道我们所应该知道的 1.9.15
我在清河市场买菜 1.9.16
漫步华尔街 1.9.17
爷爷血压飙升入院，看完之后却更不舒服了 1.9.18
生又何哀，死又何乐？庄子曰：至乐 1.9.19
生命是最基本的人权 1.9.20
老婆死了，庄子却还在唱歌，这到底怎么回事 1.9.21
读《未来简史》，思考未来 1.9.22
读《邓小平时代》 1.9.23

PanRufeng's Blog

All non-project logs, or blog articles.

CSDN

https://blog.csdn.net/prufeng

公众号

rufengp

（非技术）

Gitbook

MD to HTML

```
# Generate SUMMARY again to see if any update
> update.bat


$ node -v
v16.13.1

$ npm i -g gitbook-cli
# Fix polyfills.js:287 issue, comment out statFix()

$ gitbook -V
CLI version: 2.3.2
GitBook version: 3.2.3

# Add book.json for expandable-chapters plugins
gitbook install

# Static html menu not work
# Fix issue in _book/gitbook/theme.js, 'if(m)' -> 'if(false)', b
# Remove '()' in MD file name

gitbook serve

http://localhost:4000/
```

To Pdf

Install https://calibre-ebook.com/download_windows64

Set ebook-convert to system path: `C:\Program Files\Calibre2`

```
gitbook pdf
```

Autosys HTTPS - java.security.cert.CertificateException : subject/issuer verification failed

使用 Autosys HTTP Job 去监控服务是否正常的时候，遇到 Certificate 验证错误。

`java.security.cert.CertificateException: subject/issuer verification failed`

最后发现原因竟然是 Nginx 所使用的证书文件里有重复的元素，删掉重复的 Cert 即可。

那么，为什么 HTTP SSL 证书里会出现重复的元素？ SSL 证书里到底有些什么？

可以参考这一篇：SSL 证书 PEM 文件里有什么和要有什么

CSRF跨站请求伪造

Cross-Site Request Forgery

CSRF，Cross-Site Request Forgery，跨站请求伪造，也被称为：XSRF, One-Click attack, Sea Surf, Session Riding, Cross-Site Reference Forgery, and Hostile Linking。

CSRF是指攻击者通过某些技术手段欺骗用户，盗用其已认证的身份去执行一些恶意操作（如修改用户邮件或密码，发消息，转账和购买商品）。

CSRF漏洞与浏览器的隐式身份认证机制有关。用户在目标网页完成身份认证时，浏览器会在本地（Cookie）保存当前会话状态信息，当用户再次发送请求时，浏览器会自动携带该会话Cookie， 服务器据此判断请求来自同一用户（会话），执行请求操作。

这时假如攻击者成功诱使用户打开攻击站点，则可向目标站点发送伪造请求。由于用户已经认证过，所以浏览器将自动携带之前的会话Id，被访问网站则会认为这是用户的真实请求而去执行。

跨站是指恶意请求是由攻击站点发起，而不是用户一开始访问的可信站点。由于浏览器同源策略（Same Origin Policy）的限制，攻击者一般无法在攻击站点获取用户Cookie，而只能通过伪造请求进行攻击。

CSRF攻击场景

典型场景如下：

- 用户登录可信站点A
- 未logout的情况下访问攻击站点B
- 攻击站点B向站点A发送恶意请求

攻击实现的难点是第二步，即如何让用户在访问A的同时刚好打开B。

常见的形式有钓鱼邮件、诈骗信息、浏览器弹出窗口（浏览器劫持）、漏洞站点代码植入等。
比如你在操作网银时，刚好看到一封银行发送给你的（钓鱼）邮件，或者刚好浏览器弹出一个你感兴趣的新闻，然后你眼疾手快地点击了里面的链接。

攻击站点B不一定是恶意站点，可能只是有漏洞的普通网站，甚至是受欢迎的高流量网站。

B向A发送请求是通过技术手段自动实现。也就是说，当打开网页时，你就中招了，并不需要任何其他操作。所以尽管用户登录A的同时正好打开B的几率不大，但是黑客并不需要做什么，只是坐等收钱，因而这肯定也是一种受欢迎的攻击手段。

另外，大多数人都没有完成操作就立即退出登录的良好习惯。

CSRF请求伪造

GET

典型例子如下， `img` 的情况下不需要用户点击链接，打开窗口便自动提交请求。

```
<a href="http://bank.com/transfer.do?acct=MARIA&amount=100000">V

<img src="http://bank.com/transfer.do?acct=MARIA&amount=100000"
```

所以一般更新操作不要用Get，因为伪造成本太低。

POST

Post实现相对应的代码如下。

表单当然是不显示出来的，通常放在隐藏的iframe里。

```
<form action="<nowiki>http://bank.com/transfer.do</nowiki>" meth
<input type="hidden" name="acct" value="MARIA"/>
<input type="hidden" name="amount" value="100000"/>
<input type="submit" value="View my pictures"/>
</form>

<body onload="document.forms[0].submit()">
```

XMLHttpRequest

XMLHttpRequest方法受浏览器同源策略限制，一般不会成功，除非目标站点定义了 `Access-Control-Allow-Origin: *` 。

但是，在JavaScript的世界里，绕过同源策略的代码却很容易找到，更不用说各种各样的浏览器漏洞了。

```
<script>
function put() {
    var x = new XMLHttpRequest();
    x.open("PUT","http://bank.com/transfer.do",true);
    x.setRequestHeader("Content-Type", "application/json");
    x.send(JSON.stringify({"acct":"BOB", "amount":100}));
}
</script>
<body onload="put()">
```

CSRF防守策略

由以上分析可知，CSRF攻击之所以能奏效，主要是因为攻击者可以成功地伪造一个合法的请求，即目标操作的所有参数都可以被分析出来。

因此防范的策略主要是往请求中加入无法猜测的参数值，比如验证码和Token。

验证码

验证码是最简单有效的方法，但是因为影响用户体验，不能作为主要的解决方案。

Anti CSRF Token

CSRF Token方法与验证码类似，只是不需要用户干预。

增加一个随机生成的Token，发送给前端，前端提交表单时包含该Token，在后台验证，验证通过才是有效请求。

一个请求提交后，Token即被消耗掉，然后重新生成新Token。所以，也要考虑同时有多个请求时的并发处理问题。

注意：

- Token要足够随机，不可预测。
- Token要注意保密，不能泄露。

比如页面如果同时有XSS（跨站脚本攻击）漏洞，则攻击者完全可以读出页面的Token值，再伪造合法的请求，这一过程一般称为XSRF，与CSRF相区别。

除了加Token，也可以考虑给参数加密，或者验证访问来源（Referrer）。

JSON Web Token

假如我已经使用了JWT无状态App设计，还需不需要再引入CSRF Token？

答案是：不需要。

除了不是by request生成，JWT具有CSRF Token相同的特征，因而是安全的。

总的来说，避免使用Cookie进行身份认证，应该已经可以防范大多数的CSRF漏洞。

参考

https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)
《白帽子讲Web安全》 —— 吴翰清

Excel - VBA导入CSV为文本格式及中文乱码问题

VBA导入CSV为文本格式

Excel导入CSV时，会自动转换一些列的格式，特别是数值格式，非常烦。

找到以下VBA导入CSV为文本格式的方法，亲测可行。

先打开文件获取列数，将每一列的格式都设置为 `xlTextFormat` ，然后再赋值给 `QueryTable.TextFileColumnDataTypes` 。

```
Public Sub ImportCSVAsText()
    Dim TempWorkbook As Workbook
    Dim TempWorksheet As Worksheet
    Dim ColumnCount As Integer
    Dim FileName As Variant
    Dim ColumnArray() As Integer

    'Get the file name
    FileName = Application.GetOpenFilename(FileFilter:="All File

    If FileName = False Then Exit Sub

    Application.ScreenUpdating = False

    'Open the file temporarily to get the count of columns
    Set TempWorkbook = Workbooks.Open(FileName)
    ColumnCount = TempWorkbook.Sheets(1).Range("A1").SpecialCell
    TempWorkbook.Close SaveChanges:=False

    'Resize the array to number of columns
    ReDim ColumnArray(1 To ColumnCount)

    For i = 1 To ColumnCount
        ColumnArray(i) = xlTextFormat
    Next i

    Set TempWorkbook = Workbooks.Add
    Set TempWorksheet = TempWorkbook.Sheets(1)

    With TempWorksheet.QueryTables.Add("TEXT;" & FileName, TempW
        .FieldNames = True
        .RowNumbers = False
        .FillAdjacentFormulas = False
        .PreserveFormatting = True
        .RefreshOnFileOpen = False
        .RefreshStyle = xlInsertDeleteCells
        .SavePassword = False
        .SaveData = True
        .AdjustColumnWidth = True
        .RefreshPeriod = 0
        .TextFilePromptOnRefresh = False
        .TextFilePlatform = 1251
        .TextFileStartRow = 1
        .TextFileParseType = xlDelimited
        .TextFileTextQualifier = xlTextQualifierDoubleQuote
        .TextFileConsecutiveDelimiter = False
        .TextFileTabDelimiter = False
        .TextFileSemicolonDelimiter = False
```

```
        .TextFileCommaDelimiter = True
        .TextFileSpaceDelimiter = False
        .TextFileColumnDataTypes = ColumnArray
        .TextFileTrailingMinusNumbers = True
        .Refresh BackgroundQuery:=False
    End With

End Sub
```

中文乱码问题

修改配置 `TextFilePlatform = 65001` ，问题解决。

简单来说就是设置导入源文件的编码, 65001对应的是utf-8，编码的对应数值可以在手动导入CSV文件向导里看到。

> ## QueryTable.TextFilePlatform property (Excel)
>
> Returns or sets the origin of the text file that you are importing into the query table. This property determines which code page is used during the data import. Read/write XlPlatform.
>
> ## Remarks
>
> The default value is the current setting of the File Origin option in the Text File Import Wizard. Use this property only when your query table is based on data from a text file (with the QueryType property set to xlTextImport). If you import data by using the user interface, data from a web query or a text query is imported as a QueryTable object, while all other external data is imported as a ListObject object. If you import data by using the object model, data from a web query or a text query must be imported as a QueryTable, while all other external data can be imported as either a ListObject or a QueryTable. The TextFilePlatform property applies only to QueryTable objects.

Reference

- https://superuser.com/questions/307496/how-can-i-set-excel-to-always-import-all-columns-of-csv-files-as-text

- https://docs.microsoft.com/en-us/office/vba/api/excel.querytable.textfileplatform

HTTPS证书准备过程

使用 `keytool` 和 `openssl` 生成HTTPS证书的过程总结。

1. Generate Keystore

```
keytool -genkey -alias appssl -keyalg RSA -keysize 2048 -validit
```

2. Generate Sign Request File

```
keytool -certreq -keyalg RSA -alias appssl -file appssl.csr -key
```

3. Send Sign Request to Certificate Management Group

发给CA机构（Certificate Authority），拿回认证后的 `root cert` ，`chain cert` 和 `server cert` 。

4. Import Root Cert, Chain Cert and Server Cert.

```
keytool -list -keystore appssl.keystore -storepass password

keytool -import -alias rootcertificate -keystore appssl.keystore

keytool -import -alias Intermediate -keystore appssl.keystore -t

keytool -import -alias appssl -keystore appssl.keystore -trustca
```

5. Extract Cert and Key from JKS Keystore

```
keytool -importkeystore -srckeystore appssl.keystore -srcalias a

keytool -deststoretype PKCS12 -keystore appssl.jks.p12 -list -st

openssl pkcs12 -in appssl.jks.p12 -nokeys -clcerts -out server_s
openssl pkcs12 -in appssl.jks.p12 -nokeys -cacerts -out gs_inter
cat server_ssl.crt gs_intermediate_ca.crt >server.crt

openssl pkcs12 -nocerts -nodes -in appssl.jks.p12 -out server.ke
```

6. Setup App with Cert and Key

e.g. for Kibana

```
sudo vi /etc/kibana/kibana.yml
```

```
server.ssl.enabled: true

server.ssl.certificate: /etc/kibana/server.crt
server.ssl.key: /etc/kibana/server.key
```

相关文章

Nginx - 使用OpenSSL自签名证书配置HTTPS

JMeter测试动态参数API

使用JMeter进行压力测试时，有可能需要为每次请求传递不同的参数。

如 `/testFn/name/张三` ，需要替换掉张三，测试100次，每次请求都使用不同的名字。

这个需求可以通过 `CSV Data Set Config` ，从CSV文件读取参数变量实现。

从文件读取

HTTP Request

首先将Path从

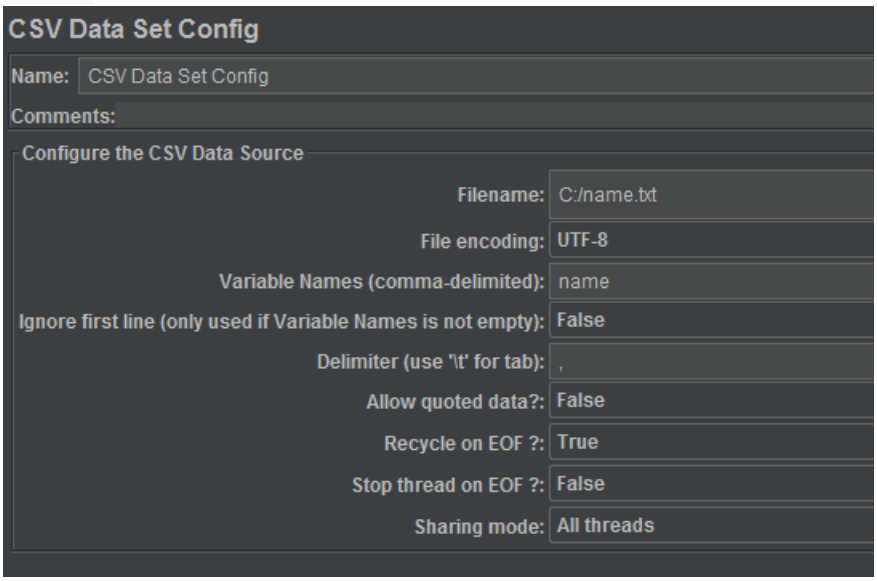
`/testFn/name/张三`

改为

`/testFn/name/${name}`

CSV Data Set Config

然后Thread Group下 `Add` -> `Config Element` -> `CSV Data Set Config` 。



当执行测试时，${name}将被从CSV文件中读出来的具体名字代替。

文件内容只包含一列，形式如下。（也可包含多列，同时支持多个变量，以逗号分隔。）

```
黄立冲
黄立川
黄立春
黄立达
黄立丹
黄立德
黄立东
黄立冬
黄立栋
黄立凡
```

CSV名字列表会被循环使用，即当测试次数(Number of Threads * Loop Count) > CSV名字数量时，名字参数将会出现重复。

其他参数变量定义方法

- JMeter内置函数
- User Defined Variable
- 从数据库读取
- 从前面请求的相应数据提取

JWT技术简单理解和实现

JSON Web Token

> JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

(https://jwt.io/introduction/)

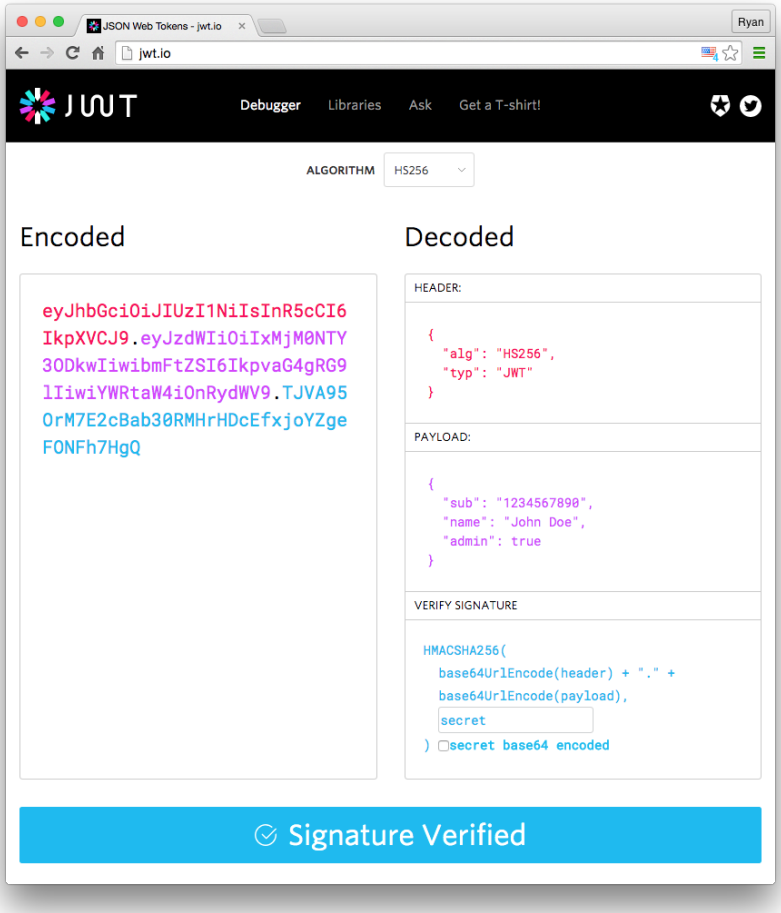
简而言之，JWT是一个JSON信息传输的开放标准，它可以使用密钥对信息进行数字签名，以确保信息是可验证和可信任的。

JWT结构

- Header
- Payload
- Signature

Header和Payload只是简单的base64编码，并没有加密，因而不能用来传递敏感内容。

Signature是加密的，接收方主要是靠这一部分进行验证。
因为加密的内容较少，运用复杂一些的算法，其效率应该也是可以接受的。



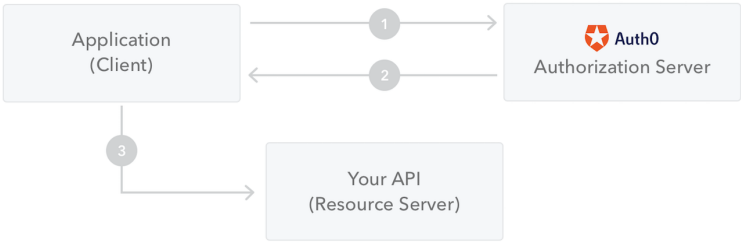
JWT用途

由JWT的结构可知，JWT并不能用来发送加密内容，而只是帮忙验证传输的内容是否中途被更改过，即所谓防篡改。

防篡改的特性使得JWT可以用来传递用户身份信息，达到服务状态保持，从而实现单点登录。

具体过程如下：

1. Client请求授权
2. Authorization Server返回access token
3. Client使用access token去访问其他服务（资源）



JWT实现要点

- Client验证成功时，生成Token
- 之后的每一个请求都要携带Token
- 后台处理每一个请求都要先验证Token

Node.js实现

- Install jsonwebtoken

https://github.com/auth0/node-jsonwebtoken

- 生成Token

```
// sign with RSA SHA256
var jwt = require('jsonwebtoken');
var privateKey = fs.readFileSync('private.key');
var token = jwt.sign({ foo: 'bar' }, privateKey, { algorithm
```

- 验证Token

```
// verify a token asymmetric
var cert = fs.readFileSync('public.pem');  // get public key
jwt.verify(token, cert, function(err, decoded) {
console.log(decoded.foo) // bar
});
```

- Decode Payload

无须签名，就可以解码Header和Payload，所以千万不要误以为它们是加密的。

```
// get the decoded payload ignoring signature, no secretOrPrivat
var decoded = jwt.decode(token);

// get the decoded payload and header
var decoded = jwt.decode(token, {complete: true});
console.log(decoded.header);
console.log(decoded.payload)
```

LVS+Keepalived+Nginx高可用负载均衡方案

负载均衡的基本思想是把单台服务器的访问分流到多台服务器，避免系统崩溃。以下是几种常见的解决方案。

1. Nginx反向代理

Nginx通过反向代理技术实现分流，可以按照设定的访问规则将请求转发给相应的后台服务器。
反向代理是指当用户访问网络上某个服务的时候，访问到的实际上是代理服务器，代理服务器转发用户请求到真正处理该业务的后台服务器，最后转发结果回来给用户。后台服务器被代理了，对用户不可见。
相比较的，正向代理则是指我们通常的翻墙过程。用户访问代理服务器，代理服务器转发请求给目标服务器，目标服务器返回结果给代理服务器，再转发回给用户。用户被代理了，对目标服务器不可见。
Nginx支持几种常见的负载调度算法：轮询、加权轮询、ip hash、fair、url hash。一致性哈希算法可保证同一个IP的请求落在同一台后端服务器，其他算法须解决不同服务器的Session共享问题。
Nginx支持健康检查，实现后端服务器的自动上下线，可以提高系统的请求成功率。如果一个服务器请求失败，可立即切换到其他服务器，直到请求成功或者最后一台服务器失败为止。

2. Keepalived+Nginx实现双机热备

当使用一台Nginx作为反向代理服务器时，这台服务器就变得异常重要，须避免出现单点故障。Keepalived是一个基于VRRP协议来实现的服务高可用方案，可以利用其来避免IP单点故障。
使用Keepalived可以轻量级地实现Nginx的双机热备。即前端使用两台服务器，当其中一台服务器发生故障时，另一台服务器接管故障机的公网虚拟IP，继续提供负载均衡服务。

3. LVS+Keepalived+Nginx增加吞吐量和稳定性

Nginx的请求转发是基于HTTP层的，所有流量都要经过Nginx。当后端服务器规模庞大时，Nginx的网络带宽就会成为一个瓶颈。
这时可引入LVS负责第一层负载，通过LVS将请求路由到后端若干台Nginx上，Nginx再转发给内部其中一台应用服务器进行处理，最后Nginx直接将结果转发给用户，无需再经过LVS服务器。
以上过程实际上用到的是LVS/DR（Linux Virtual Server via Direct

Routing）技术，它通过改写请求报文的MAC地址，将请求发送到真实服务器，而真实服务器将响应直接返回给客户。
LVS还提供的另外两种IP负载均衡技术是：VS/NAT（Virtual Server via Network Address Translation）和VS/TUN（Virtual Server via IP Tunneling）。
LVS通过MAC地址改写和网络地址转换等方式来同步转发请求，主要是基于网络层和链路层，因而效率非常高。
引入LVS后，Nginx变成了集群，Keepalived主要用来保证LVS的高可用，建立LVS双机热备。LVS 可以对后端Nginx进行健康检查（不能重新发送请求）。LVS也支持多种负载调度算法，如：RR, WRR, LC, WLC, LBLC, LBLCR, DH, SH。

4. DNS和CDN网络优化

DNS（Domain Name System）中一个域名可对应多个IP，因而可通过DNS轮询来实现负载均衡。同时DNS智能解释可保证返回给用户的IP离用户尽可能的近。
还可以进一步按照业务系统拆分二级域名，对于每一个业务系统建立独立的负载均衡层。
CDN（Content Delivery Network）主要用于缓存图片、多媒体文件等各种静态文件资源。将这些静态资源保存到离用户更近的各个网络节点，可以大大提高用户访问效率和节省服务器外网出口带宽。

Nginx - Basic Authentication

Install Nginx

Download package from https://nginx.org/download/nginx-1.16.1.tar.gz

```
tar zxvf nginx-1.16.1.tar.gz
cd nginx-1.16.1/

sudo ./configure --with-http_stub_status_module --with-http_ssl_
sudo make
sudo make install
```

Install dependencies if need. (e.g. pcre pcre-devel, zlib zlib-develo, penssl, openssl-devel)

```
Configuration summary
  + using system PCRE library
  + OpenSSL library is not used
  + using system zlib library
```

Install done.

```
$ sudo nginx -V
nginx version: nginx/1.16.1
built by gcc 4.8.5 20150623 (Red Hat 4.8.5-36) (GCC)
built with OpenSSL 1.0.2k-fips  26 Jan 2017
TLS SNI support enabled
configure arguments: --with-http_stub_status_module --with-http_
```

Startup Nginx

```
ln -s /usr/local/nginx/sbin/nginx /usr/bin/nginx
sudo nginx

$ ps aux|grep nginx
root     14112  0.0  0.0  20556   612 ?        Ss   10:24   0:00
nobody   14113  0.0  0.0  23092  1388 ?        S    10:24   0:00
admin    14127  0.0  0.0 112716   976 pts/2    S+   10:24   0:00
$ curl localhost
<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.16.1</center>
</body>
</html>
```

Install httpd-tools

To generate the `auth_basic_user_file`.

```
yum install httpd-tools


sudo htpasswd -b /usr/local/nginx/conf/passwd <username> <passwo
```

The password file looks as below.

```
$ cat test.pwd
test:$apr1$y0scTAxi$EHurcBgQzTak3nwFAHfyC/
test1:$apr1$4ddNEKao$jqKyZAnqvjYKlceeBQiQN.
```

Reload Nginx Configure

```
sudo vi /usr/local/nginx/conf/nginx.conf
sudo nginx -s reload
```

1. Change nobody to root for test (or grant right to nobody on passwd)
2. Add auth_basic configure

```
#user  nobody;
user   root;


    server {
        listen       80;
        server_name  192.168.1.88;

        auth_basic "Basic Auth";
        auth_basic_user_file /usr/local/nginx/conf/passwd;

        #charset koi8-r;

        #access_log  logs/host.access.log  main;

        location / {
            proxy_pass http://localhost:5601;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection 'upgrade';
            proxy_set_header Host $host;
            proxy_cache_bypass $http_upgrade;
            #root   html;
            #index  index.html index.htm;
        }
```

Change Kibana Configure

```
sudo vi config/kibana.yml
```

Change from `0.0.0.0` to `localhost`.

```
#server.host: "0.0.0.0"
server.host: "localhost"
```

修改后重启。

想保留原来5601端口的链接不禁用，这里可以用具体IP，相应的 `proxy_pass` 也要改。

Verification

需要登录才能进入Kibana。

http://192.168.1.88

Reference

http://nginx.org/en/docs/http/ngx_http_auth_basic_module.html
https://httpd.apache.org/docs/2.4/programs/htpasswd.html

nginx: [emerg] cannot load certificate SSL: error:0200100D:system library:fopen:Permission denied:fopen

换了SSL Cert之后，Nginx无法启动，发生如下错误。

> nginx: [emerg] cannot load certificate "/etc/nginx/ssl/test.crt": BIO_new_file() failed (SSL: error:0200100D:system library:fopen:Permission denied:fopen('/etc/nginx/ssl/test.crt','r') error:2006D002:BIO routines:BIO_new_file:system lib)

- 确认使用 `sudo` 运行。
- 确认该文件可以 `cat` 打开。
- 确认文件权限与原Cert一致。
- `nginx -t` successful.

查了一下，原来是 SELinux 的问题。

`SELinux`，即 Security Enhanced Linux，是一种基于委任式存取控制（Mandatory Access Control, MAC）的权限控制系统，比通常所理解的用户自主式存取控制 （Discretionary Access Control, DAC）更严格。

DAC 主要根据程序的拥有者和文件的rwx权限来决定用户可以进行的操作，但 root 可以控制一切，777文件可以被任何人操作。

MAC 可以通过规则限定特定的程序只能操作指定的文件，这样就算是 root 运行的程序，也不能随意访问其他不相关的文件了。

`SELinux` 默认使用 `targeted` 模式。

```
getenforce

sestatus -v
```

检查 Audit log 可以发现，文件不能访问的原因，是程序与目标文件的 `scontext` (Security Context) 不一致。

Nginx 使用的是 `httpd_t`，自己copy的文件使用的是 `user_home_t`。

导致这个问题的操作是，先在 `home` 下创建了文件，然后再 `mv` 到当前目录，而 `scontext` 不会因复制移动而修改。

```
$ sudo ausearch -m avc -ts today

time->Thu Apr  9 12:00:42 2020
type=PROCTITLE msg=audit(1586404842.411:635786): proctitle=2F757
type=SYSCALL msg=audit(1586404842.411:635786): arch=c000003e sys
type=AVC msg=audit(1586404842.411:635786): avc:  denied  { read
```

查看测试 Cert 与原来版本的区别。

```
$ sudo ls -lrtZ /etc/nginx/ssl
-rw-r--r--. root root system_u:object_r:httpd_config_t:s0 origin
-rw-r--r--. root root system_u:object_r:httpd_config_t:s0 origin
-rw-r--r--. root root unconfined_u:object_r:user_home_t:s0 test.
-rw-r--r--. root root unconfined_u:object_r:user_home_t:s0 test.
```

使用 `restorecon` 让文件恢复正确的 SELinux type。

```
$ sudo restorecon -v -R /etc/nginx/ssl/test.*
restorecon reset /etc/nginx/ssl/test.crt context unconfined_u:ob
restorecon reset /etc/nginx/ssl/test.key context unconfined_u:ob
$ sudo ls -lrtZ /etc/nginx/ssl
-rw-r--r--. root root system_u:object_r:httpd_config_t:s0 origin
-rw-r--r--. root root system_u:object_r:httpd_config_t:s0 origin


-rw-r--r--. root root unconfined_u:object_r:httpd_config_t:s0 te
-rw-r--r--. root root unconfined_u:object_r:httpd_config_t:s0 te
```

可以使用 `semanage` 查询和修改默认的目录安全性本文，此即 `restorecon` 恢复的所谓正确 SELinux type 的来源。

```
$ sudo semanage fcontext -l|grep etc/nginx
/etc/nginx(/.*)?                                   all files
```

重启 Nginx， 成功！

Nginx - LDAP Authentication

比Nginx - Basic Authentication稍微高级一点点的LDAP认证。

Install http_auth_request_module

```
sudo ./configure --with-http_stub_status_module --with-http_ssl_
sudo make
sudo make install

$ sudo nginx -V
nginx version: nginx/1.16.1
built by gcc 4.8.5 20150623 (Red Hat 4.8.5-36) (GCC)
built with OpenSSL 1.0.2k-fips  26 Jan 2017
TLS SNI support enabled
configure arguments: --with-http_stub_status_module --with-http_
```

Install python-ldap

I download and install offline.

```
# sudo yum install python-ldap
sudo rpm -hvi python-ldap-2.4.15-2.el7.x86_64.rpm

# sudo yum install openldap-devel
# cyrus-sasl-devel is dependency package
sudo rpm -hvi cyrus-sasl-devel-2.1.26-23.el7.x86_64.rpm
sudo rpm -hvi openldap-devel-2.4.44-21.el7_6.x86_64.rpm
```

Nginx Configure

Download package from https://github.com/nginxinc/nginx-ldap-auth

Use sample configure.

```
cp /usr/local/nginx/conf/nginx.conf /usr/local/nginx/conf/nginx.
cp nginx-ldap-auth.conf /usr/local/nginx/conf/nginx.conf
sudo nginx -t
sudo nginx -s reload
```

The listen port is 8081 in the sample configure. We need to change/open the port and use correct LDAP configure for test.

Startup LDAP Daemon

Run python below to start the daemon and sample app.

- nginx-ldap-auth-daemon.py
- backend-sample-app.py

Can use the following bash to start the daemon.

```
nginx-ldap-auth-daemon-ctl-rh.sh start
nginx-ldap-auth-daemon-ctl-rh.sh stop
```

nginx: [emerg] the "ssl" parameter requires ngx_http_ssl_module

Nginx加ssl_certificate配置后报这个错。

将相应模块安装回来即可。

```
$ sudo nginx -s reload
nginx: [emerg] the "ssl" parameter requires ngx_http_ssl_module
```

修改前

```
$ sudo nginx -V
nginx version: nginx/1.16.1
built by gcc 4.8.5 20150623 (Red Hat 4.8.5-36) (GCC)
configure arguments:
```

重新编译

```
sudo ./configure --with-http_stub_status_module --with-http_ssl_
sudo make
```

Copy ./objs/nginx

```
sudo cp /usr/local/nginx/sbin/nginx /usr/local/nginx/sbin/nginx.
sudo ls -l /usr/local/nginx/sbin/
sudo cp ./objs/nginx /usr/local/nginx/sbin/
```

（特别再试了一下，直接 `sudo make install` 也是可以的，并没有覆盖原有的配置文件。）

重启

```
ps -aux|grep ngi
sudo kill -9 <PID>
/usr/local/nginx/sbin/nginx
```

安装成功

```
$ sudo nginx -V
nginx version: nginx/1.16.1
built by gcc 4.8.5 20150623 (Red Hat 4.8.5-36) (GCC)
built with OpenSSL 1.0.2k-fips  26 Jan 2017
TLS SNI support enabled
configure arguments: --with-http_stub_status_module --with-http_
```

Nginx - 使用OpenSSL自签名证书配置HTTPS

OpenSSL生成证书

- 创建Key
- 创建签名请求（CSR）
- 移除口令
- 签署证书（或发给CA机构认证）```

  ### Generate an RSA key

  openssl genrsa -des3 -out Server.key 1024

Creating Certificate Signing Requests

需要输入组织信息，留空输入点(.)，而非直接回车（用缺省值）

openssl req -new -key Server.key -out Cert.csr

移除口令

cp Server.key Server.key.org openssl rsa -in Server.key.org -out Server.key

Signing Your Own Certificates

openssl x509 -req -days 365 -in Cert.csr -signkey Server.key -out Cert.crt

```
# Nginx SSL配置
确保`http_ssl_module`已安装。

否则会报错：[nginx: [emerg] the "ssl" parameter requires ngx_http
```

$ sudo nginx -V nginx version: nginx/1.16.1 built by gcc 4.8.5 20150623 (Red Hat 4.8.5-36) (GCC) built with OpenSSL 1.0.2k-fips 26 Jan 2017 TLS SNI support enabled configure arguments: --with-http_stub_status_module --with-http_ssl_module

```
加ssl配置。
```

sudo vi /usr/local/nginx/conf/nginx.conf

```conf
        listen       443 ssl;
        server_name  192.168.1.88;

        ssl_certificate /usr/local/nginx/conf/Cert.crt;
        ssl_certificate_key /usr/local/nginx/conf/Server.key;
```

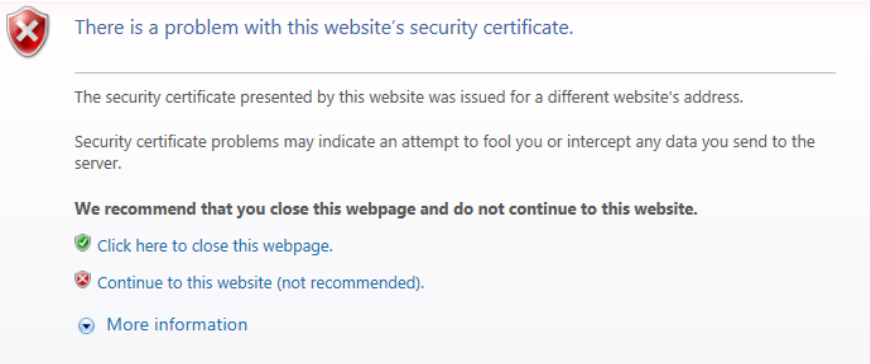
开443端口

```
sudo firewall-cmd --list-all

sudo firewall-cmd --zone=public --add-port=443/tcp --permanent


sudo firewall-cmd --zone=public --remove-port=443/tcp --permanen
sudo firewall-cmd --reload
```

测试

提示是一个有问题的证书（因为是自己签名的）。

https://192.168.1.88



参考

https://www.feistyduck.com/library/openssl-cookbook/online/ch-openssl.html https://nginx.org/en/docs/http/ngx_http_ssl_module.html

```
$EmailFrom = “from_email_id”
$EmailTo = “to_email_id”
$Subject = “Subject”
$Body = “Hello User”
$SMTPServer = “smtp.gmail.com”
$SMTPClient = New-Object Net.Mail.SmtpClient($SmtpServer, 587)
$SMTPClient.EnableSsl = $true
$SMTPClient.Credentials = New-Object System.Net.NetworkCredentia
$SMTPClient.Send($EmailFrom, $EmailTo, $Subject, $Body)
```

Tomcat报错

27-Jan-2022 14:09:07.450 信息 [main] org.apache.coyote.AbstractProtocol.start 开始协议处理句柄["https-jsse-nio-8443"] 27-Jan-2022 14:09:07.459 严重 [main] org.apache.catalina.util.LifecycleBase.handleSubClassException 无法启动组件[Connector[AJP/1.3-8009]] org.apache.catalina.LifecycleException: 协议处理器启动失败 at org.apache.catalina.connector.Connector.startInternal(Connector.java:1075) at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:183) at org.apache.catalina.core.StandardService.startInternal(StandardService.java:449) at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:183) at org.apache.catalina.core.StandardServer.startInternal(StandardServer.java:927) at org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:183) at org.apache.catalina.startup.Catalina.start(Catalina.java:772) at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method) at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62) at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43) at java.lang.reflect.Method.invoke(Method.java:498) at org.apache.catalina.startup.Bootstrap.start(Bootstrap.java:345) at org.apache.catalina.startup.Bootstrap.main(Bootstrap.java:476) Caused by: java.lang.IllegalArgumentException: AJP连接器配置secretRequired="true",但是属性secret确实空或者空字符串，这样的组合是无效的。 at org.apache.coyote.ajp.AbstractAjpProtocol.start(AbstractAjpProtocol.java:270) at org.apache.catalina.connector.Connector.startInternal(Connector.java:1072) ... 12 more

vi conf/server.xml

直接设置 `secretRequired=""` 可以解决，不过这种办法似乎不应该提倡，因为default是true。

不如加一个密码， `secret='secret'`

secret

Only requests from workers with this secret keyword will be accepted. The default value is null. This attribute must be specified with a non-null, non-zero length value unless secretRequired is explicitly configured to be false. If this attribute is configured with a non-null, non-zero length value then the workers must provide a matching value else the request will be rejected irrespective of the setting of secretRequired.

secretRequired

If this attribute is true, the AJP Connector will only start if the secret attribute is configured with a non-null, non-zero length value. This attribute only controls whether the secret attribute is required to be specified for the AJP Connector to start. It does not control whether workers are required to provide the secret. The default value is true. This attribute should only be set to false when the Connector is used on a trusted network.

26-Jan-2022 09:52:13.920 警告 [main] org.apache.catalina.webresources.Cache.getResource 无法将位于[/WEB-INF/classes/*]的资源添加到Web应用程序[/helloworld]的缓存中，因为在清除过期缓存条目后可用空间仍不足 - 请考虑增加缓存的最大空间。

vi tomcat\conf\context.xml

```
<Resources cachingAllowed="true" cacheMaxSize="102400" />
```

cacheMaxSize

The maximum size of the static resource cache in kilobytes. If not specified, the default value is 10240 (10 megabytes). This value may be changed while the web application is running (e.g. via JMX). If the cache is using more memory than the new limit the cache will attempt to reduce in size over time to meet the new limit. If necessary, cacheObjectMaxSize will be reduced to ensure that it is no larger than cacheMaxSize/20.

cacheObjectMaxSize

Maximum size of the static resource that will be placed in the cache. If not specified, the default value is 512 (512 kilobytes). If this value is greater than cacheMaxSize/20 it will be reduced to cacheMaxSize/20. This value may be changed while the web application is running (e.g. via JMX).

Solution Architect是做什么的：流程，角色描述，责任和成果

关于Solution Architect，感觉这篇文章写的比较清晰，故简单翻译整理一下。

不太字斟字酌，因为有些直译感觉怪怪的，比如“谁是解决方案架构师”，实在有些反应不过来。

https://www.altexsoft.com/blog/engineering/solution-architect-role/

Who is Solution Architect: Processes, Role Description, Responsibilities, and Outcomes

根据2017 Project and Portfolio Management Landscape总结，Planview调查的49％的公司在过去12个月内都遭遇过项目失败。

在快速变化的技术环境中，组织需要转变其流程和系统，以满足新兴的业务需求。这种数字化转型需要特定的专业知识和一系列实践，以使技术解决方案与业务目标保持一致。

解决方案架构属于在任何技术解决方案开发以前执行的最重要实践之一。在本文中，我们将讨论解决方案架构是什么，描述解决方案架构师的角色，并解释如何采用此实践帮助解决业务问题。

解决方案架构是设计，描述和管理特定业务问题解决方案工程的实践。

解决方案架构师是负责领导这种实践，并介绍特定解决方案整体技术愿景的人。

虽然可以在内部进行管理，但也有公司提供解决方案架构咨询的服务。

我们将在本文中讨论这些术语以及它们所包含的多个方面涵义。

企业架构vs解决方案架构与技术架构

企业架构

企业架构主要专注于构建复杂的企业生态系统和解决高层战略问题。企业架构定义了业务架构的战略方向，从而使我们可以了解支持该架构所需的技术设施。

解决方案架构

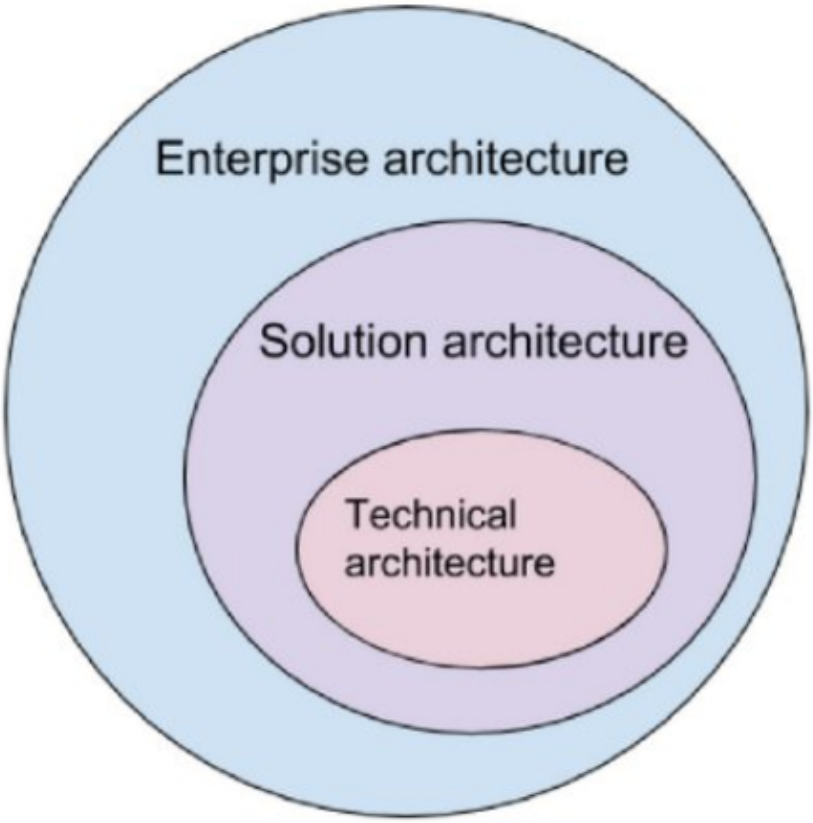
解决方案架构（SA）是一个复杂的过程，具有许多子流程，弥补了业务问题和技术解决方案之间的差距。它包括：

- 寻找可能解决现有业务问题的最佳技术解决方案。
- 向项目利益相关者描述软件的结构，特征，行为及其他方面。
- 定义功能，阶段和解决方案需求。
- 提供定义，管理和交付解决方案的规范。

技术架构

技术架构主要负责工程问题和软件架构。

有些组织还专门添加了基础架构。对于在内部维护复杂硬件基础架构以支持运营的组织而言，此角色非常重要。



解决方案架构描述了在特定解决方案中如何使用不同的业务组件，信息和技术架构。 由于解决方案架构更侧重于解决特定业务问题的细节和解决方案技术，因此它为企业架构和技术架构之间提供了一个通道，后者仅关注于技术问题。 对于公司来说，一个有效的方法是将企业架构活动与解决方案和技术架构活动相结合。

解决方案架构的主要流程

精心构建的解决方案架构可帮助团队按时，按预算开发产品，并确保解决其旨在解决的问题。解决方案架构实际执行了哪些任务？

将解决方案与企业环境相匹配

通常，公司已经拥有操作系统，信息环境和集成要求。解决方案架构要确保新系统适合现有的企业环境。为此，解决方案架构师必须了解业务模型的所有部分如何协同工作，包括流程，操作系统和应用程序架构。了解这些流程后，他们才能够设计出最适合该环境的特定解决方案。

满足所有利益相关者的需求

软件产品开发最重要的挑战之一是满足利益相关者的需求。通常，一个产品有许多利益相关者，既有技术专家又有非技术专家。解决方案架构的目标是确保将他们所有的需求都考虑在内。而且我们必须定期向利益相关者通报产品开发流程，成本和预算。这些任务通常由解决方案架构师来完成，他将技术项目细节转换为管理和其他非技术人员都可以理解的语言。

考虑项目约束

每个项目都有其局限性，通常称为项目约束。包括：

- 技术
- 风险
- 范围
- 成本
- 质量
- 时间
- 资源

例如，产品所采用的技术必须符合其特定模块的需求。 项目范围是定义软件特定目标，任务，特性和功能的文档的一部分。 所有的项目也都有预算分配。

虽然这些方面本身就是约束，但每个方面都有其自身的局限性。 解决方案架构师必须了解所有这些约束，进行比较，然后做出一系列技术和管理决策，以便将这些限制与项目目标相协调。

选择项目技术栈

创建解决方案架构的一个重要部分是选择产品开发技术。技术架构策略将直接取决于所选的技术栈。许多不同的实践都与平台，编程语言和工具相关。这时解决方案架构的职责就是去找到一种最适合特定项目的实

践，而不仅仅是普通的技术评估和比较。

满足非功能性需求

所有软件项目都必须满足一些体现其系统特性的非功能性需求，也称为质量属性。虽然非功能性需求的确切列表取决于每个产品的复杂性，但最常见的是产品的安全性，性能，可维护性，可扩展性，可用性和可靠性。解决方案架构师的职责之一是分析所有非功能性需求，并确保进一步的产品工程能够满足这些需求。

由于我们已经讨论了解决方案架构采用的高级目标，现在我们将其细分为具体职责和基础技能组合。

解决方案架构师职责

解决方案架构师的工作重点是解决方案级决策，及其对整体业务目标和结果的影响分析。 解决方案架构师的职责直接源于以下实践中的过程：

- 分析技术环境
- 分析企业特性
- 分析和记录需求
- 设置协作框架
- 创建解决方案原型
- 参与技术选型
- 控制解决方案开发
- 支持项目管理

虽然这些职责中的大部分都需要解决方案架构师的领导角色，但他们只是支持PM活动，以确保资源，风险识别和计划与解决方案的最终目标保持一致。

解决方案架构师角色描述

SOLUTION ARCHITECT ROLE	
Responsibilities	✓ Analyzing technology environment ✓ Analyzing enterprise specifics ✓ Analyzing requirements ✓ Setting collaboration framework ✓ Creating a solution prototype ✓ Participating in technology selection ✓ Solution development control ✓ Project management support
Experience	✓ At least eight years of working experience in one or multiple IT areas ✓ IT infrastructure and cloud development ✓ Engineering and software architecture design ✓ Business analysis ✓ DevOps ✓ Project and product management
Personal Attributes	✓ Excellent communication skills ✓ Deep analytical skills ✓ Project and resource management skills ✓ Work collaboratively ✓ Influence and negotiate

如果您已经意识到解决方案架构在您的项目中的重要性，那么下一步就是为该角色寻找和雇用员工。 我们已经考虑了一个好的解决方案架构师必须具备的核心技能。

技术背景和经验

为了向管理和工程团队提供技术建议，解决方案架构师必须具备在一个或多个IT领域至少具有八年工作经验的技术背景，包括但不限于：

- IT架构，基础架构和云开发
- 工程和软件架构设计
- 业务分析
- DevOps
- 项目和产品管理

出色的沟通技巧

解决方案架构师能力的一个重要部分是沟通。考虑到这一角色需要与利益相关方进行谈判，了解各方的需求，管理风险和产品交付，缺乏沟通技巧可能会造成真正的瓶颈。该职位需要与企业和软件架构师，业务分析师及项目团队密切合作。因此，经验丰富的解决方案架构师都应该具有倾听，建议，共情和解说的能力。

深厚的分析能力

设计解决方案需要了解业务的不同部分如何协同工作。架构师必须认识到公司战略，并实现所有那些定义了公司如何达到其战略目标的业务流程。架构师还必须了解技术细节。所以，解决方案架构师需要不断地处理分析工作，并且在不同的业务层之间来回跳转。

项目和资源管理技能

虽然解决方案架构师不直接参与项目管理，但是不可避免地也要考虑项目的截止日期和给定资源。解决方案架构师必须能够决断哪些解决方案有利，哪些解决方案在特定情况下毫无价值。他们必须关注业务成果，并了解如何在给定的时间和资源下实现这些成果。

公司什么时候需要解决方案架构咨询

如果集成新软件系统的过程不是系统化的，那么技术咨询组织可以将解决方案架构引入公司结构。最重要的一点是，没有必要为每个项目都配置一个解决方案架构师。如果您只是实现一个已经验证过在类似环境中可行的技术模块，那么您就不需要解决方案架构师。一旦软件工程项目变得更加复杂，并且具有相当多的流程和风险，最好考虑解决方案架构咨询服务。

让我们来看看需要引进解决方案架构咨询的案例：

您不知道哪种解决方案最适合企业生态系统。 您需要在现有企业架构和特定项目之间建立链接。解决方案架构师将确保该解决方案符合您公司环境的标准。例如，如果您将许多现有的后台流程迁移到云，解决方案架构师将负责分析多种方案，并结合技术，人力和财务资源给出最好的建议。

您运行数字转型项目。 数字化转型要求企业重新考虑如何为客户提供价值或最终提供什么。如果不将技术和业务视角联系起来，就没有办法回答这些问题，而这正是解决方案架构师工作的核心。

您面临很多风险。 这种情形下解决方案架构师是必要的，比如项目存在许多技术风险，可能出现不确定的需求，必须同时实施多个产品，基础技术尚未得到批准。

您必须向投资者推销产品路线图。 如果您必须向投资者和利益相关者展示一个未来的产品，解决方案架构师将给您建议符合产品需求的技术，并以清晰平实的业务术语说明其推理论证过程。

您需要在利益相关者和工程师之间建立沟通。 解决方案架构师将有助于填补技术和非技术专家之间的沟通差距。此外，通过与项目管理团队合作，解决方案架构师可以覆盖产品与利益相关方需求不一致的风险，并确保解决方案符合既定目标。

许多团队都参与了这个项目。 您可能有一个大型项目，需要有人来管理设计师，技术或业务架构师团队，以创造高质量的技术项目成果。

结语

无论您是否真正采用这种实践，解决方案架构都是任何IT项目的基础。通过有意识地引入解决方案架构，您可以构建一种框架，使技术，资源和技能，与既定的业务目标保持一致。

在考虑解决方案架构之前，我们建议您探索结合了解决方案和企业层面的企业架构框架。

传统上，中小型企业既不考虑解决方案架构的实践，也不考虑各自的角色。解决方案架构问题的处理是在产品团队中的其他角色之间委派和共享的。对于具有可预测结果的小型项目来说，这是一个可行的选择。但是，如果您有一个复杂的企业架构，将多个软件产品引入一个正常运行的生态系统，您需要完全重建您的遗留软件，或制定一套战略性技术决策，那么，需要一个专业的专家，就是显而易见的了。

发现git bash里curl命令也可以用，无需安装其他工具就可以用来简单测试RESTful API，非常方便。

1. GET

```
curl localhost:3000

curl -X GET localhost:3000

include header:
curl -i localhost:3000

verbose info:
curl -v localhost:3000

curl -v -X GET localhost:3000/users
curl -v -X GET localhost:3000/users/1
```

2. POST

```
curl -v -d "id=1&name=user1&age=18" localhost:3000/users
```

Note: Unnecessary use of -X or --request, POST is already inferred.

```
curl -v -X POST -d "id=1&name=user1&age=18" localhost:3000/users

curl -v -H "Content-type: application/json" -X POST -d '{"id":"1

curl -H "Content-type: application/json" -X POST -d '{"id":"1","
```

3. PUT

```
curl -v -X PUT -d "id=1&name=user1&age=127" localhost:3000/users

curl -v -H "Content-type: application/json" -X PUT -d '{"name":"
```

4. DELETE

```
curl -v -X DELETE localhost:3000/users/1
```

How to Test Web Service Performance With JMeter

Download & Install

Download and unzip. Run bin/jmeter.bat, will see the UI to new test plan.

https://jmeter.apache.org/

Test Plan

Thread Group

Action to be taken after a Sampler error

- Continue
- Start Next Thread Loop
- Stop Thread
- Stop Test
- Stop Test Now

`Stop Test Now` will stop the whole test immediately. Any current samplers are interrupted if possible.

Thread Properties

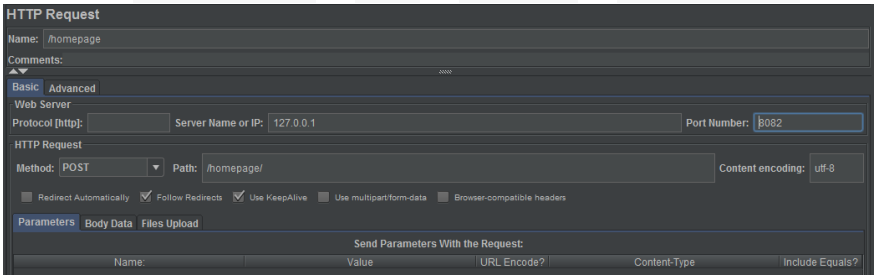
- Number of Threads (users)
- Ramp-Up Period (in seconds)
- Loop Count
- Scheduler

- Duration (seconds)
- Startup delay (seconds)

Ramp-Up Period means the time Jemter need to start the `Number of Threads` .
`Loop Count` is for each thread, that means if we have 5 threads, with loop count 2, the service will get 10 requests totally.

HTTP Request

Right click `Thread Group` -> `Add` -> `Sampler` -> `HTTP Request` .



Input protocol, IP, port, method, path, etc.

Insert data into `Parameters` or `Body Data` .
If body data is required for JSON or XML, add appropriate `Content-Type` in `HTTP Header Manager` .

HTTP Header Manager

Right click `HTTP Request` -> `Add` -> `Config Element` -> `HTTP Header Manager` .

- Test SOAP

Content-Type text/xml

- Test RESTful With JSON

Content-Type application/json

Kick Start

Right click the thread group name and click `Start` will run tests in this thread group.

Click `Start` button in the top of the screen will run all thread group tests.

Listener

When you just start the test and want to see the response, to double confirm if the request is feedback correctly, you can add `Listener`.

Right click the HTTP request -> `Add` -> `Listener` -> `View Result Tree`.

You can see many other Listeners also. e.g.

- Summary Report
- Aggregate Report
- Graph Result

After you have confirmed your service feedback is okay, you can right click `View Result Tree` to disable it, if you concern it may impact the overall test performance.

Test in Command Line

```
jmeter -n -t my_test.jmx -l log.jtl
```

Generate HTML report

```
jmeter.bat -n -t cms-restful-homepage.jmx -l log.jtl -e -o repor
```

Report dashboard overview:

https://jmeter.apache.org/usermanual/generating-dashboard.html

Test Strategy

- Setup measurement baseline
- Integrate Jmeter Into CI Tool (Jenkins/Bamboo/SonarQube)
- Create Test Plan for DEV, SIT and UAT

Questions:

- Test one service or multiple services per run?
- How many requests per service?
- How many requests per second?
- How to measure it is good or not?
- When to run(trigger) the performance test?

重构 - Java程序太多if else怎么办

有人问，我的程序太多if else了，怎么办？

让发过来看看，长这样。

```
public void execute(boolean ba, boolean bb, boolean bc) {
    if (ba && bb && bc) {
        response = callApiByChannelName(getChannelNameForThr
    } else if (ba && bb && !bc) {
        response = callApiByChannelName(getChannelNameForTwo
    } else if (ba && !bb && bc) {
        response = callApiByChannelName(getChannelNameForTwo
    } else if (!ba && bb && bc) {
        response = callApiByChannelName(getChannelNameForTwo
    } else if (ba && !bb && !bc) {
        response = callApiOfChannelA(aChannel, xLoad, userIn
    } else if (!ba && bb && !bc) {
        response = callApiOfChannelB(bChannel, xLoad, userIn
    } else if (!ba && !bb && bc) {
        response = callApiOfChannelC(cChannel, xLoad, userIn
    }
}
```

我说你不是看过设计模式吗？
看过，可是还是不知道怎么办？

程序设计的原则简单来说就两点，首先是怎么把无关的东西分开，然后是怎么把共同的东西抽出来。

好吧，看你也听不明白，帮你分析一下。

先理解业务逻辑再写代码

一看你这个条件就是一个排列组合嘛，你
看 `getChannelNameForThree()` 这个方法是从三个中选一个是吧？

然后 `getChannelNameForTwo()` 是从两个中选一个，剩下的不用选。

上面这四个 `callApiByChannelName()` 最终调用的实际上还是下面那三个方法，对吧？

而且我相信三个中选一个两个中选一个的逻辑应该是一样的，但是现在你的ifelse，却被写成了这个策略的一部分。

其实应该把它抽出来，先决定好调用哪个接口，然后再调用它，这样就根本不需要这么多ifelse了。

所以问题不在于ifelse，而是你没有真正理解业务逻辑。

代码中多几个ifelse也并不是多严重的问题，真正的问题是，如果再加一个新成员来组合，该怎么办？

另一个问题就是，这个代码再放几天，估计连你自己也搞不清楚它到底是要做什么的了。

说完他去检查业务逻辑去了，等他检查完，我的模拟重构代码也基本上写完了。

第一步优化结果如下，已经解决了ifelse多且难看的问题。

```
public void step1() {
    int i = getTheRightChannel();
    if (i == 1) {
        response = callApiOfChannelA(aChannel, xLoad, userIn
    } else if (i == 2) {
        response = callApiOfChannelB(bChannel, xLoad, userIn
    } else {
        response = callApiOfChannelC(cChannel, xLoad, userIn
    }
}
```

他检查完之后的回复在我意料之中，不过他却兴奋得像发现新大陆。我很理解这种心情。

使用工厂来创建对象

下一步要考虑的是，假如要再加一个Channel，要怎么办？

按照当前的程序，需要增加一个条件，再写一个方法。如果方法比较复杂，程序必然越来越臃肿。

但观察这几个方法，只是调用的远程接口的细节有所不同，所以可以抽象成一个统一的接口。

```
public void step2() {
    int i = getTheRightChannel(  );
    Channel channel;
    if (i == 1) {
        channel = new ChannelA();
    } else if (i == 2) {
        channel = new ChannelB();
    } else {
        channel = new ChannelC();
    }
    response = channel.callApi(xLoad, userInfo);
}
```

接着把 `Channel` 的创建放到一个统一的地方。这种方法被称为简单工厂模式。

```
public void step3() {
    int i = getTheRightChannel();
    ChannelSimpleFactory channelSimpleFactory = new ChannelS
    Channel channel = channelSimpleFactory.createChannel(i);
    response = channel.callApi(xLoad, userInfo);
}
```

现在若要再增加 `Channel` ，以上代码已经无需修改了，只需新建一个 `Channel` 实现类，并在工厂里增加创建该类的逻辑。

个人觉得，大多数的重构，到这一步就可以了。程序逻辑已相当清晰，扩展所需的修改也已非常简单和可预见。

进一步的重构，需要考虑成本收益比，所以最好是等到下一次需求变更的时机。

修改只是扩展

当然，简单工厂模式还是不符合OCP原则 (Open-Closed Principle)。

> Open for extention, closed for modification.

程序应该设计成容易扩展的，且不需要修改的。换句话说，最好的程序是，当你修改的时候，你只是在扩展。

针对当前这个业务，要达到这种程度，可以把不同种类的 `Channel` 保存在配置里，再通过 `List` 或 `Map` 加载出来。

这样，每当增加新 `Channel` 时，只需要写新的 `Channel` 实现类，再把它添加到配置列表里就可以。

以下代码中 `Map` 的设置只是演示加载过程，实际上要放在配置里。

```
public void step4() {
    int i = getTheRightChannel();
    ChannelSimpleFactory channelSimpleFactory = new ChannelS
    // should be in the configure
    Map map = new HashMap<Integer, Channel>();
    map.put(1, new ChannelA());
    map.put(2, new ChannelB());
    map.put(3, new ChannelC());
    channelSimpleFactory.setChannelMap(map);
    //
    Channel channel = channelSimpleFactory.createChannel(i);
    response = channel.callApi(xLoad, userInfo);
}
```

策略变了怎么办

这样修改之后，似乎很完美了，但其实还有一个地方，甚至比以上这步更值得优化。

那就是怎样选出合适 `Channel` 的策略， `getTheRightChannel()` ，这是第二可能变更需求的地方。

想想哪天老板突然走过来说，发布之前再帮忙改改这个吧，不要随机选 `Channel` 了，给加个权重吧。

（此处省略一万字……）

于是又要加配置，加方法，加ifelse，老路重走。

不过现在起码还是可以修改的，看看最初的代码，根本无从下手！

可以考虑加一个 `ChannelStrategy` 接口，当更换策略时，只需要新建一个子类 `ChannelWeightStrategy` 注入 `ChannelSimpleFactory` 就可以了。

```
public void step5() {
    ChannelSimpleFactory channelFactory = new ChannelSimpleF
    // or by configure
    ChannelStrategy strategy = new ChannelWeightStrategy();
    channelFactory.setStrategy(strategy);
    //
    Channel channel = channelFactory.newChannel();
    response = channel.callApi(xLoad, userInfo);
}
```

策略模式理解起来比较简单，不再赘述，不同项目细节也可能不同，不如直接看代码。

https://github.com/prufeng/hellowork/tree/master/src/main/java/pan/rufeng/pattern/refactor/ifelsefactory

太多类会不会不太好

最后还有一个奇怪的问题，就是，怎么看你加了那么多的类，会不会不太好？

嗯，这个，不知道怎么说，你去看看JDK或Spring里的源码吧，看看都长什么样，或者用代码工具再扫扫自己的，对比一下。

好啦，重构适可而止，不然其他项目交不了货啦！

Anaconda多版本Python切换

安装Anaconda3，就自带了Python3.

这时可以用Conda建一个Python2的env。

```
conda create --name py2 python=2.7
activate py2
python -V
deactivate

conda create --name py3 python=3.7
activate py3
python -V

C:\Users\pan>conda env list
# conda environments:
#
base                  *  C:\swdtools\anaconda3
py2                      C:\swdtools\anaconda3\envs\py2
py3                      C:\swdtools\anaconda3\envs\py3
```

Git Command Alias

这段时间在用命令行，忘记哪里抄了一份这样的Git快捷命令，也加了一些自定义部分，都习惯了，记录一下，防丢。

~/.bash_profile

```
alias gpm='git push origin master'
alias ...=../..
alias ....=../../..
alias .....=../../../..
alias ......=../../../../..
alias 1='cd -'
alias 2='cd -2'
alias 3='cd -3'
alias 4='cd -4'
alias 5='cd -5'
alias 6='cd -6'
alias 7='cd -7'
alias 8='cd -8'
alias 9='cd -9'
alias _=sudo
alias afind='ack -il'
alias d='dirs -v | head -10'
alias g=git
alias ga='git add'
alias gaa='git add --all'
alias gapa='git add --patch'
alias gau='git add --update'
alias gb='git branch'
alias gba='git branch -a'
alias gbd='git branch -d'
alias gbda='git branch --no-color --merged | command grep -vE "^
alias gbl='git blame -b -w'
alias gbnm='git branch --no-merged'
alias gbr='git branch --remote'
alias gbs='git bisect'
alias gbsb='git bisect bad'
alias gbsg='git bisect good'
alias gbsr='git bisect reset'
alias gbss='git bisect start'
alias gc='git commit -v'
alias 'gc!'='git commit -v --amend'
alias gca='git commit -v -a'
alias 'gca!'='git commit -v -a --amend'
alias gcam='git commit -a -m'
alias 'gcan!'='git commit -v -a --no-edit --amend'
alias 'gcans!'='git commit -v -a -s --no-edit --amend'
alias gcb='git checkout -b'
alias gcd='git checkout develop'
alias gcf='git config --list'
alias gcl='git clone --recursive'
alias gclean='git clean -fd'
alias gcm='git checkout master'
alias gcmsg='git commit -m'
```

```
alias 'gcn!'='git commit -v --no-edit --amend'
alias gco='git checkout'
alias gcount='git shortlog -sn'
alias gcp='git cherry-pick'
alias gcpa='git cherry-pick --abort'
alias gcpc='git cherry-pick --continue'
alias gcs='git commit -S'
alias gcsm='git commit -s -m'
alias gd='git diff'
alias gdca='git diff --cached'
alias gdct='git describe --tags `git rev-list --tags --max-count
alias gdt='git diff-tree --no-commit-id --name-only -r'
alias gdth='git diff-tree --no-commit-id --name-only -r HEAD'
alias gdw='git diff --word-diff'
alias gf='git fetch'
alias gfa='git fetch --all --prune'
alias gfo='git fetch origin'
alias gg='git gui citool'
alias gga='git gui citool --amend'
alias ggpull='git pull origin $(git_current_branch)'
alias ggpur=ggu
alias ggpush='git push origin $(git_current_branch)'
alias ggsup='git branch --set-upstream-to=origin/$(git_current_b
alias ghh='git help'
alias gignore='git update-index --assume-unchanged'
alias gignored='git ls-files -v | grep "^[[:lower:]]"'
alias git-svn-dcommit-push='git svn dcommit && git push github m
alias gk='\gitk --all --branches'
alias gke='\gitk --all $(git log -g --pretty=%h)'
alias gl='git pull'
alias gl1='git log -1'
alias gl2='git log -2'
alias glg='git log --stat --oneline'
alias glgg='git log --graph'
alias glgga='git log --graph --decorate --all'
alias glgm='git log --graph --max-count=10'
alias glgp='git log --stat -p'
alias glo='git log --oneline --decorate'
alias globurl='noglob urlglobber '
alias glog='git log --oneline --decorate --graph'
alias gloga='git log --oneline --decorate --graph --all'
alias glol='git log --graph --pretty='\''%Cred%h%Creset -%C(yell
alias glola='git log --graph --pretty='\''%Cred%h%Creset -%C(yel
alias glp=_git_log_prettily
alias glum='git pull upstream master'
alias gm='git merge'
alias gmom='git merge origin/master'
alias gmt='git mergetool --no-prompt'
```

```
alias gmtvim='git mergetool --no-prompt --tool=vimdiff'
alias gmum='git merge upstream/master'
alias gp='git push'
alias gpd='git push --dry-run'
alias gpoat='git push origin --all && git push origin --tags'
alias gpristine='git reset --hard && git clean -dfx'
alias gpsup='git push --set-upstream origin $(git_current_branch
alias gpu='git push upstream'
alias gpv='git push -v'
alias gr='git remote'
alias gra='git remote add'
alias grb='git rebase'
alias grba='git rebase --abort'
alias grbc='git rebase --continue'
alias grbi='git rebase -i'
alias grbm='git rebase master'
alias grbs='git rebase --skip'
alias grep='grep  --color=auto --exclude-dir={.bzr,CVS,.git,.hg,
alias grh='git reset HEAD'
alias grhh='git reset HEAD --hard'
alias grm='git rm --cache'
alias grmv='git remote rename'
alias grrm='git remote remove'
alias grset='git remote set-url'
alias grt='cd $(git rev-parse --show-toplevel || echo ".")'
alias gru='git reset --'
alias grup='git remote update'
alias grv='git remote -v'
alias gsb='git status -sb'
alias gsd='git svn dcommit'
alias gsi='git submodule init'
alias gsh='git show --stat --oneline HEAD'
alias gshh='git show --stat --oneline'
alias gsps='git show --pretty=short --show-signature'
alias gsr='git svn rebase'
alias gs='git status'
alias gss='git status -s'
alias gst='git status'
alias gsta='git stash save'
alias gstaa='git stash apply'
alias gstc='git stash clear'
alias gstd='git stash drop'
alias gstl='git stash list'
alias gstp='git stash pop'
alias gsts='git stash show --text'
alias gsu='git submodule update'
alias gts='git tag -s'
alias gtv='git tag | sort -V'
```

```
alias gunignore='git update-index --no-assume-unchanged'
alias gunwip='git log -n 1 | grep -q -c "\-\-wip\-\-" && git res
alias gup='git pull --rebase'
alias gupv='git pull --rebase -v'
alias gwch='git whatchanged -p --abbrev-commit --pretty=medium'
alias gwip='git add -A; git rm $(git ls-files --deleted) 2> /dev
alias history='fc -l 1'
alias l='ls -lah'
alias la='ls -lAh'
alias ll='ls -lh'
alias ls='ls --color=tty'
alias lsa='ls -lah'
alias md='mkdir -p'
alias please=sudo
alias po=popd
alias pu=pushd
alias rd=rmdir
alias which-command=whence
```

error: RPC failed; HTTP 500 curl 22 The requested URL returned error: 500 Internal Server Error

```
$ git push -u origin master
Enumerating objects: 46, done.
Counting objects: 100% (46/46), done.
Delta compression using up to 8 threads.
Compressing objects: 100% (44/44), done.
error: RPC failed; HTTP 500 curl 22 The requested URL returned e
fatal: The remote end hung up unexpectedly
Writing objects: 100% (46/46), 8.53 MiB | 7.21 MiB/s, done.
Total 46 (delta 8), reused 0 (delta 0)
fatal: The remote end hung up unexpectedly
Everything up-to-date
```

```
git config --global http.postBuffer 524288000
```

Get Started

Clone

```
git clone https://github.com/prufeng/hellowork.git

git remote show origin
git remote get-url origin
git remote remove origin
git remote add origin https://github.com/prufeng/hellowork.git
```

Store/Unset Password

```
git config --global credential.helper store
git config --global --unset credential.helper
```

Changes

Commit

```
echo "Helloworld" >>README.md
git add .
git commit -m "First commit"
git commit -m 'Multiple lines comment

1. Fix bug 1
2. Fix bug 2'

git push
```

Status

```
git status
git status -s
```

List Changed Files in Commit

```
git show --name-only --oneline HEAD
git show --stat --oneline HEAD

git log --stat --oneline HEAD
git log --name-only --oneline -2
git diff-tree --name-only -r <commit id>
```

Amend

```
git commit --amend --only
git commit --amend --only -m 'xxxxxxx'
git commit --amend --author "New Authorname <authoremail@mydomai
```

Add Change to Last Commit

```
git add .
git commit --amend
```

Add Some Change in File

```
git add -p <file>
```

Discard changes in File

```
git checkout <file>
git checkout HEAD <file>
```

Undo Add

```
git reset
```

Reset Commit

```
git reset HEAD~
git reset HEAD~1
git reset <commit id> <--soft|mixed|hard>
```

Remove File

```
git rm --cached <file>
```

Remove File from Last Commit

```
git checkout HEAD~ <file>
git add .
git commit --amend
```

Find back Hard Reset

```
git reflog -2
git reset --hard <commit id>
```

Stash

```
git stash
git pull
git stash pop

git push
```

Cherry Pick

```
git cherry-pick <commit id>
```

Branch

Pull Wrong Branch

```
git reflog
git reset --hard <commit id>
```

Align Local with Remote

```
git reset --hard origin/master
```

Checkout From Remote Branch

```
git fetch --all
git checkout --track origin/<branch>
git checkout -b <branch> origin/<branch>
```

Delete Branch

```
git branch -D <branch>
git push origin --delete <branch>
```

Restore Deleted Branch

```
git reflog
git checkout -b <new-branch>
git reset --hard <commit id>
```

Fallback Rebase and Merge

```
git reset --hard ORIG_HEAD
```

New Branch Without Local Changes

```
git checkout -b current_feature
git add .
git commit -m "Add current feature"

git checkout master
git checkout -b new_feature_without_current_feature
```

Push Commit from New Branch

```
git checkout new_feature_without_current_feature
git cherry-pick <commit id>
git push origin HEAD:master
```

Python基础

发现Python都忘得差不多了，简单复习一下。

Install

- anaconda
- conda

Input

```
help(input)

name = input('what is your name? ')
```

Modules and Imports

Can use dir() to check the namespace difference.

```
import math
import math as fun
from math import pi, tan
from math import pi as pie, tan as tangent
```

Data Type

- int
- float
- bool

Basic Math Functions

math pi e isinf isnan factorial log exp pow sqrt cos acos degrees radians acosh asinh

Sequence Types

- str
- byte

- bytearray
- list
- tuple
- slicing

slice[start, stop, step]

Collection & Mapping

- range
- set
- dict

Loops & Conditions

- while
- for
- if

SSL证书PEM文件里有什么和要有什么

Privacy-Enhanced Mail (PEM) 本来是研究来给邮件加密的，现在主要成为一种约定俗成的文件格式，用于存储和发送加密密钥，证书或其他数据。

简单理解就是格式如下这种文件。

```
-----BEGIN XXX-----
BASE64
-----END XXX-----
```

JKS Keystore 转化为 PEM 的过程

在HTTPS SSL证书准备过程里介绍过，我们生成了证书请求发给 CA 后，会收到一个 JKS Keystore 。

JKS Keystore 转化为 PEM 文件过程如下。

```
keytool -importkeystore -srckeystore appssl.jks -destkeystore ap

keytool -deststoretype PKCS12 -keystore appssl.jks.p12 -list -st


openssl pkcs12 -in appssl.jks.p12 -nokeys -clcerts -out client.c
openssl pkcs12 -in appssl.jks.p12 -nokeys -cacerts -out ca.crt
cat client.crt ca.crt >server.crt

openssl pkcs12 -nocerts -nodes -in appssl.jks.p12 -out server.ke
```

`nokeys` 表示不包含 Private Key， `clcerts` 输出 Client Cert， `cacerts` 输出 Root Cert 和 Intermediate Cert。

`-nocerts -nodes` 即不包含 Cert 和不加密。

实际上 `clcerts` 和 `cacerts` 两个文件合起来和下面这句输出一致。

```
openssl pkcs12 -in appssl.jks.p12 -nokeys -out server.crt
```

使用 `keytool` 也可以直接从jks导出PEM，不过需要按alias逐个导出。

```
keytool -list -exportcert -keystore appssl.jks  -alias 'appssl'
```

使用 `openssl` 导出的时候，还会添加如下Attributes。

```
Bag Attributes
    friendlyName: app
    localKeyID: 54 69 6D 65 20 41 45 48 49 45 44 42 44 40 42 49
subject=...app.dummy.com
issuer=...Entrust Certification Authority - L1M
```

而且 Root Cert 和 Intermediate Cert 会出现重复，因为它们在 JKS Keystore 里本来就是重复的。

JKS Keystore 里有什么

查看 JKS Keystore ， 可以发现里面包含3个实体，一个 PrivateKeyEntry 和两个 trustedCertEntry。

但 PrivateKeyEntry 里 `Certificate chain length: 3` ， 即包含了另外两个 Cert，于是使用 Openssl转化以后 Root Cert 和 Intermediate Cert 就出现了重复。

```
$ keytool -list -keystore appssl.jks -rfc

Keystore type: jks
Keystore provider: SUN

Your keystore contains 3 entries

Alias name: appssl
Creation date: Apr 18, 2020
Entry type: PrivateKeyEntry
Certificate chain length: 3
Certificate[1]:
-----BEGIN CERTIFICATE-----
BASE64
-----END CERTIFICATE-----
Certificate[2]:
-----BEGIN CERTIFICATE-----BASE64
-----END CERTIFICATE-----
Certificate[3]:
-----BEGIN CERTIFICATE-----BASE64
-----END CERTIFICATE-----


*******************************************
*******************************************


Alias name: cn=entrust certification authority - intermediate
Creation date: Apr 18, 2020
Entry type: trustedCertEntry

-----BEGIN CERTIFICATE-----
BASE64
-----END CERTIFICATE-----


*******************************************
*******************************************


Alias name: cn=entrust root certification authority
Creation date: Apr 18, 2020
Entry type: trustedCertEntry

-----BEGIN CERTIFICATE-----
BASE64
-----END CERTIFICATE-----
```

```
****************************************
****************************************
```

SSL 证书 PEM 文件里要有什么

特地做了下试验。

使用 Openssl 导出的 PEM 当然是可以了，里面包含了5个 PEM 。

删掉重复的 Root Cert 和 Intermediate Cert 之后，表现也正常。

最后，只留下 Client Cert，浏览器里表现正常，应用程序里测试会出错。

```
javax.net.ssl.SSLHandshakeException: PKIX path building failed:
```

我的简单理解是，浏览器可以联网查找认证链，应用程序不行。

相关规范

https://tools.ietf.org/html/rfc1421
https://tools.ietf.org/html/rfc7292
https://tools.ietf.org/html/rfc7468

Upgrade Python2.7.5 to Python2.7.16

Python has already been installed in Linux by default, but the version is usually not up to date.

Install Python 2.7.16

Download from https://www.python.org/ftp/python/2.7.16/Python-2.7.16.tgz

Specify prefix and use altinstall in case overwriting the default Python.

```
tar zxvf Python-2.7.16.tgz
cd Python-2.7.16

sudo ./configure --prefix=/usr/local
sudo make
sudo make altinstall
```

```
mv /usr/bin/python /usr/bin/python2.7.5
ln -s /usr/local/bin/python2.7 /usr/bin/python
```

We can find the different version `python` under `usr/bin`.

```
$ python -V
Python 2.7.16
$ python2.7.5 -V
Python 2.7.5
$ python2.7 -V
Python 2.7.16
$ python2 -V
Python 2.7.5
```

替换旧 `python` 的操作，如非真的必要，不建议。

因为替换后发现一些系统命令会出问题，范围似乎也无法预知，有些风险。

No module named yum

Yum does not work after Python upgrade.

```
$ sudo yum install zlib
There was a problem importing one of the Python modules
required to run yum. The error leading to this problem was:

   No module named yum

Please install a package which provides this module, or
verify that the module is installed correctly.

It's possible that the above module doesn't match the
current version of Python, which is:
2.7.16 (default, Aug 30 2019, 16:44:16)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)]

If you cannot solve this problem yourself, please go to
the yum faq at:
  http://yum.baseurl.org/wiki/Faq
```

```
sudo vi /usr/bin/yum
```

Change `#! /usr/bin/python` to `#! /usr/bin/python2`

ImportError: No module named urlgrabber.grabber

```
Downloading packages:
Traceback (most recent call last):
  File "/usr/libexec/urlgrabber-ext-down", line 22, in <module>
    from urlgrabber.grabber import \
ImportError: No module named urlgrabber.grabber
Traceback (most recent call last):
  File "/usr/libexec/urlgrabber-ext-down", line 22, in <module>
    from urlgrabber.grabber import \
ImportError: No module named urlgrabber.grabber

sudo vi /usr/libexec/urlgrabber-ext-down
```

Change `#! /usr/bin/python` to `#! /usr/bin/python2`

vi Python-2.7.16/Modules/Setup.dist

```
#zlib zlibmodule.c -I$(prefix)/include -L$(exec_prefix)/lib -lz
zlib zlibmodule.c -I$(prefix)/include -L$(exec_prefix)/lib -lz
```

Install Python again.

```
sudo ./configure --prefix=/usr/local
sudo make
sudo make altinstall
```

ImportError: No module named _ssl

```
sudo yum install openssl-devel

sudo vi /Modules/Setup
```

```
#_socket socketmodule.c timemodule.c
_socket socketmodule.c timemodule.c
```

Install Python again.

VMWare虚拟机CentOS联网：主机能SSH连虚拟机，虚拟机能上网

这两天玩一下VMWare，配置虚拟机网络的时候，发现网上很多误导，还是稍微记录一下，以免反复被带坑里。

搜了几个方法，不是ping不进去就是ping出来，又或者上不了网。正确的方法却非常简单。

VMWare版本：`12.5.9 build-7535481`

CentOS版本：

```
# cat /etc/redhat-release
CentOS Linux release 7.8.2003 (Core)
```

选择桥接模式

选择桥接模式，看到自定义那里有得选，所以我就直接选了，效果应该是一样的。

主机上其他网络设置，都不需要改！

物理机网络配置

供虚拟机参考。

```
>ipconfig

以太网适配器 本地连接:
   IPv4 地址 . . . . . . . . . . . . : 192.168.31.95
   子网掩码  . . . . . . . . . . . . : 255.255.255.0
   默认网关. . . . . . . . . . . . . : 192.168.31.1
```

虚拟机网络配置

虚拟机IP地址要与物理机在同一网段。

```
# vi /etc/sysconfig/network-scripts/ifcfg-ens33
```

```
ONBOOT=yes
BOOTPROTO=static
IPADDR=192.168.31.191
NETMASK=255.255.255.0
GATEWAY=192.168.31.1
DNS1=192.168.31.1
ZONE=public
```

```
systemctl restart network
```

CentOS SSH配置

CentOS 7.8 sshd默认是开的，所以不用管。

```
# yum install net-tools
# netstat -anpt|grep sshd
tcp        0      0 0.0.0.0:22              0.0.0.0:*
tcp        0     28 192.168.31.191:22       192.168.31.95:49909
tcp        0      0 192.168.31.191:22       192.168.31.95:52616
```

Firewall默认也是开的。可以先关掉试试。 如果可以，把端口打开。

```
systemctl stop firewalld
systemctl start firewalld
firewall-cmd --zone=public --add-port=22/tcp --permanent
firewall-cmd --list-ports
firewall-cmd --reload
```

Win10不用虚拟机也可以装Linux（Ubuntu）

想在Windows下玩Linux，下了个虚拟机。

突然想起Win10好像有虚拟机，找一下，果然！

可是，这个 `适用于Linux的Windows子系统` ，是什么鬼？

难道是自带Linux？

果断试一下，选择，需要重启。

打开Microsoft Store，搜索Ubuntu。

看到有Ubuntu 20和Ubuntu 18，20提示不兼容，那就安装18好了。

应该是系统版本号的限制，不行可能要升级自身系统。

安装完成之后，直接打开，是一个命令行窗口，像模像样。

需要设置用户名和密码，但是不能是root。

没有默认的root用户，su不过去。

需要先这样设置root用户密码。

```
$ sudo passwd
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

设置完后root就出来了。

Win10磁盘在 `/mnt/c/` 。

```
$ ll /mnt/
total 0
drwxr-xr-x 1 root root 4096 Dec  9 19:52 ./
drwxr-xr-x 1 root root 4096 Dec  9 19:52 ../
drwxrwxrwx 1 feng feng 4096 Dec 10 22:57 c/
```

另外试了一下，是可以联网的。

```
# curl baidu.com
<html>
<meta http-equiv="refresh" content="0;url=http://www.baidu.com/"
</html>
```

貌似没有什么bug，明天装点软件试试。

Define Mutilple Pipelines For Logstash

Logstash有一个对初学者来说很大的坑，就是安装之后直接运行，它将默认只使用一个pipeline，把conf.d下面的配置文件一次全加载了。

如果不同配置文件的输入输出有冲突，比如缺少必要的条件判断，就会发生意想不到的结果。

使用多管道，势在必行。

Default Pipeline

pipeline.yml

默认的pipeline配置是这样子的。

```
# This file is where you define your pipelines. You can define m
# For more information on multiple pipelines, see the documentat
#   https://www.elastic.co/guide/en/logstash/current/multiple-pi

- pipeline.id: main
  path.config: "/etc/logstash/conf.d/*.conf"
```

conf.d/app.conf

然后配置里需要一大堆ifelse，来区分来自不用的应用。

比如单说处理Filebeat的log，就需要在filter里根据不同的关键字分别进行处理。

```
input {
  beats {
    port => 5044
  }
}

filter {
  if [fields][type] == "aaaa" {
    grok {
        ...
    }
    date {
        ...
    }
  } else if [fields][type] == "bbbb" {
    grok {
        ...
    }
    date {
        ...
    }
  } else if [fields][type] == "cccc" {
    grok {
        ...
    }
    if "_grokparsefailure" in [tags] {
      drop{}
    }
    date {
        ...
    }
  }
}

output {
  if [fields][type] in ["aaaa", "bbbb", "cccc"] {
    elasticsearch {
      hosts => ["http://192.168.0.888:9200"]
      index => "%{[@metadata][beat]}-%{[fields][type]}-%{+YYYY.M
    }
  }
}
```

输出其实也一样，不过这里用了些适配的技巧。

增加其他类型，一样要增加ifelse的判断。

Multiple Pipelines

pipeline.yml

在pipeline里按App分流。

每个App分别定义一个pipeline，再放到独立的配置文件里。

```
- pipeline.id: beats
  config.string: |
    input { beats { port => 5044 } }
    output {
      if [fields][type] == "aaaa" { pipeline { send_to => aaaa_a
      } else if [fields][type] == "bbbb" { pipeline { send_to =>
      } else if [fields][type] == "cccc" { pipeline { send_to =>
    }

- pipeline.id: aaaa
  path.config: "/etc/logstash/conf.d/{aaaa,beats_out}.conf"

- pipeline.id: bbbb
  path.config: "/etc/logstash/conf.d/{bbbb,beats_out}.conf"

- pipeline.id: cccc
  path.config: "/etc/logstash/conf.d/{cccc,beats_out}.conf"
```

配置文件路径支持适配的写法。每个App分别加载两个文件。

这里这样处理主要是因为他们的output配置刚好可以是一样的。

beats_out.conf

虽然共享同一个配置，但是因为在不同管道，所以不需要加ifelse也不会混乱。

```
output {
  elasticsearch {
    hosts => ["http://192.168.0.888:9200"]
    index => "%{[@metadata][beat]}-%{[fields][type]}-%{+YYYY.MM}
  }
}
```

aaaa.conf

从上游管道地址接收输入。

直接处理log不需要再判断。

```
input { pipeline { address => aaaa_addr } }

filter {
  grok {
      ...
  }
  if "_grokparsefailure" in [tags] {
    drop{}
  }
  date {
      ...
  }
}
```

调试

可以先用最简单的例子来调试，一个一个添加。

```
- pipeline.id: upstream
  config.string: input {file { path => "/etc/logstash/iiii.log"
- pipeline.id: downstream
  config.string: input { pipeline { address => myVirtualAddress
```

我就是太心急了，直接刷刷一大堆，结果总是有错。

半天才发现，原来只是删除ifelse的时候，有些文件没有删掉多余的缩进，空格有些混乱，肉眼又很难看出来。

ELK - ElastAlert最大的坑

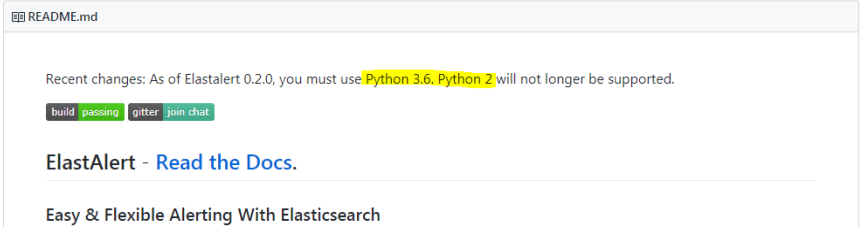
我可以讲粗口吗！

官网说Python2.7！

实际要用Python3.6！

https://elastalert.readthedocs.io/en/latest/running_elastalert.html

https://github.com/Yelp/elastalert



Elasticsearch: Validation Failed: 1: this action would add [2] total shards, but this cluster currently has [1000]/[1000] maximum shards open

Elasticsearch突然就写不进去了，看日志，shards有1000的限制。

Dev Tool

```
{
  "error" : {
    "root_cause" : [
      {
        "type" : "runtime_exception",
        "reason" : "Failed to persist transform configuration"
      }
    ],
    "type" : "runtime_exception",
    "reason" : "Failed to persist transform configuration",
    "caused_by" : {
      "type" : "validation_exception",
      "reason" : "Validation Failed: 1: this action would add [2
    }
  },
  "status" : 500
}
```

elasticsearch.log

```
Caused by: org.elasticsearch.common.ValidationException: Validat
        at org.elasticsearch.indices.ShardLimitValidator.validat
        at org.elasticsearch.cluster.metadata.MetadataCreateInde
        at org.elasticsearch.cluster.metadata.MetadataCreateInde
        at org.elasticsearch.cluster.metadata.MetadataCreateInde
        at org.elasticsearch.cluster.metadata.MetadataCreateInde
        at org.elasticsearch.action.admin.indices.create.AutoCre
        at org.elasticsearch.cluster.ClusterStateUpdateTask.exec
        at org.elasticsearch.cluster.service.MasterService.execu
        at org.elasticsearch.cluster.service.MasterService.calcu
        at org.elasticsearch.cluster.service.MasterService.runTa
        ... 10 more
```

权宜之计，当然是改大了。

```
PUT /_cluster/settings
{
  "transient": {
    "cluster": {
      "max_shards_per_node":2000
    }
  }
}
```

碰巧一起重启时，Kibana可能会因为这个错启动不了。

```
[Failed to create internal index mappings[org.elasticsearch.comm
```

但其实Elasticsearch还是可以访问的，用命令行即可。

```
curl localhost:9200/_cluster/health?pretty -uelasticuser:changem

curl -X PUT "localhost:9200/_cluster/settings?pretty" -uelasticu
{
  "transient": {
    "cluster": {
      "max_shards_per_node":2000
    }
  }

}
'
```

以上命令只是临时更改配置，下次重启之后还是会失败，要持久化，可将transient改为persistent，不过注意这并没有从根本上解决shards太多的问题。

Elasticsearch JVM OutOfMemory, Kibana: [circuit_breaking_exception] [parent] Data too large

Kibana打开就报错，其实也算是Kibana的bug，看到已经有人起了Issue，就算Elasticsearch出现Out of memory，但也不至于整个网页都死了嘛。

测试版本：7.9.0

> {"statusCode":500,"error":"Internal Server Error","message":"[parent] Data too large, data for [] would be [1044896078/996.4mb], which is larger than the limit of [1020054732/972.7mb], real usage: [1044895496/996.4mb], new bytes reserved: [582/582b], usages [request=136/136b, fielddata=8487/8.2kb, in_flight_requests=582/582b, model_inference=0/0b, accounting=56798692/54.1mb]: [circuit_breaking_exception] [parent] Data too large, data for [] would be [1044896078/996.4mb], which is larger than the limit of [1020054732/972.7mb], real usage: [1044895496/996.4mb], new bytes reserved: [582/582b], usages [request=136/136b, fielddata=8487/8.2kb, in_flight_requests=582/582b, model_inference=0/0b, accounting=56798692/54.1mb], with { bytes_wanted=1044896078 & bytes_limit=1020054732 & durability=\"PERMANENT\" }"}

解决办法：

Elasticsearch JVM内存加大试试？默认的太小了！

```
vi /etc/elasticsearch/jvm.options
```

```
-Xms1g
-Xmx1g
```

ELK - Failed to start Elasticsearch. elasticsearch.service start operation timed out. Terminating. Can't move to started state when closed

升级到最新版后发现Elasticsearch启动不了，总是time out。

```
journalctl -u elasticsearch
```

```
systemd[1]: Starting Elasticsearch...
systemd[1]: elasticsearch.service start operation timed out. Ter
systemd-entrypoint[24309]: uncaught exception in thread [main]
systemd-entrypoint[24309]: java.lang.IllegalStateException: Can'
systemd-entrypoint[24309]: at org.elasticsearch.common.component
systemd-entrypoint[24309]: at org.elasticsearch.common.component
systemd-entrypoint[24309]: at org.elasticsearch.node.Node.start(
systemd-entrypoint[24309]: at org.elasticsearch.bootstrap.Bootst
systemd-entrypoint[24309]: at org.elasticsearch.bootstrap.Bootst
systemd-entrypoint[24309]: at org.elasticsearch.bootstrap.Elasti
systemd-entrypoint[24309]: at org.elasticsearch.bootstrap.Elasti
systemd-entrypoint[24309]: at org.elasticsearch.cli.EnvironmentA
systemd-entrypoint[24309]: at org.elasticsearch.cli.Command.main
systemd-entrypoint[24309]: at org.elasticsearch.cli.Command.main
systemd-entrypoint[24309]: at org.elasticsearch.bootstrap.Elasti
systemd-entrypoint[24309]: at org.elasticsearch.bootstrap.Elasti
systemd-entrypoint[24309]: For complete error details, refer to
systemd[1]: Failed to start Elasticsearch.
systemd[1]: Unit elasticsearch.service entered failed state.
systemd[1]: elasticsearch.service failed.
```

解决办法是增加服务启动时间。

缺省值是90s。

```
$ systemctl show elasticsearch | grep ^Timeout
TimeoutStartUSec=1min 30s
TimeoutStopUSec=0
```

修改为180。

```
$ systemctl edit elasticsearch.service

[Service]
TimeoutStartSec=180
```

文件实际被保存到以下路径，会覆盖服务原本的缺省值。

```
cat /etc/systemd/system/elasticsearch.service.d/override.conf
```

修改之后，TimeoutStartUSec变为3min。

```
$ systemctl daemon-reload
$ systemctl show elasticsearch | grep ^Timeout
TimeoutStartUSec=3min
TimeoutStopUSec=0
```

这里有个Linux的小坑。 `TimeoutStartUSec` 配置在文件里应该为 `TimeoutStartSec` ，不知道为什么这么无聊显示的时候要加多个 `U` 。

Elasticsearch什么时候使用Data Transform和Rollup Jobs

问题场景

两个log，一个是信息发送记录，一个是信息接收记录，都load到同一个index里，想要通过查询这个index来监控信息是否发送成功，亦即判断发送log里出现的信息id，能不能在接收log里找到，找到就算成功，否则发送告警邮件。 我想到的方法是，用painless script，分别按log类型不同aggregate之后，再循环判断发送的id是不是都能在接收日志里找到，不过painless script功能比较简单，测试比较麻烦。

Elastic Architect推荐用Data Transforms。

Data Transforms

试用一下，发现所谓Data Transforms，其实是定时将一个index的数据转换到另一个新的index。当我们想在统计数据的基础上再做展示时，就非常有用了，特别是它不但包括了常规的Aggretion，还可以Group by。暂时还不太清楚Group by的背后是怎么实现的。

不过数据转换有一个转换时点的问题，就是在某一时刻，部分数据已经转换到新index，可是如果源index还在持续更新，新的数据只能等到下次转换再统计，这样就导致实际想要的数据被截断了，得到的不是我们想要的结果。虽然理论上来说数据不可避免会在某一时刻被截断，但直接在源数据上查询，新的查询就会包含最新的数据，得出正确的结果，但转换不一样，下次再转换，并不会修正之前已经转换的记录。这样看起来，在固定的时间用来做统计和备份是有价值的，但用做实时告警似乎就不太严谨了，可能会误报偏多。

Elastic对Data Transforms的目标表述如下：

> Use transforms to pivot existing Elasticsearch indices into summarized or entity-centric indices.

Transform API

```
GET _transform/<job_id>
GET _transform/<job_id>/_stats
POST _transform/<job_id>/_start
POST _transform/<job_id>/_stop
post _transform/<job_id>/_update
```

Data Rollup Jobs

相比之下，Rollups功能则明确表示，它是用来备份数据的，使我们在源数据删除以后，仍能获得历史数据的概况。

有些数据长期保留的价值偏低，这时，保留一个类似统计快照的信息或许就已经足够了。

> Summarize and store historical data in a smaller index for future analysis.

/job/

```
PUT /_rollup/job/<job_id>: Create a rollup job
GET /_rollup/job: List rollup jobs
GET /_rollup/job/<job_id>: Get rollup job details
POST /_rollup/job/<job_id>/_start: Start a rollup job
POST /_rollup/job/<job_id>/_stop: Stop a rollup job
DELETE /_rollup/job/<job_id>: Delete a rollup job
```

/data/

```
GET /_rollup/data/<index_pattern>/_rollup_caps: Get Rollup Capab
GET /<index_name>/_rollup/data/: Get Rollup Index Capabilities
```

//

```
GET /<index_name>/_rollup_search: Search rollup data
```

ELK - Elasticsearch启动不了: Could not create the Java Virtual Machine

提供一个思路。

Elasticsearch启动不了

```
Nov 04 15:45:10 appserver01 elasticsearch[25283]: Invalid -Xlog
Nov 04 15:45:10 appserver01 elasticsearch[25283]: Error: Could n
Nov 04 15:45:10 appserver01 elasticsearch[25283]: Error: A fatal
```

去看log发现整个folder都不见了（被人删了？）。

重建，grant right to elasticsearch就OK了。

```
/var/log/elasticsearch
```

ELK - Elasticsearch权威指南里的简单搜索例子

Elasticsearch权威指南是指官网的《Elasticsearch：权威指南》。

例子跟官网不一样，是因为类型（Type）这种用法现在已经被抛弃了，所以缺省类型都是 `_doc` 。

建立索引

```
PUT /employee/_doc/1
{
    "first_name" : "John",
    "last_name" :  "Smith",
    "age" :        25,
    "about" :      "I love to go rock climbing",
    "interests": [ "sports", "music" ]
}
PUT /employee/_doc/2
{
    "first_name" :  "Jane",
    "last_name" :   "Smith",
    "age" :         32,
    "about" :       "I like to collect rock albums",
    "interests":  [ "music" ]
}

PUT /employee/_doc/3
{
    "first_name" :  "Douglas",
    "last_name" :   "Fir",
    "age" :         35,
    "about":        "I like to build cabinets",
    "interests":  [ "forestry" ]
}
```

检索文档

```
GET employee/_doc/1

#index info
GET employee
```

轻量搜索

```
GET employee/_search
GET employee/_search?q=last_name:Smith
```

查询表达式

```
GET employee/_search
{
  "query": {
    "match": {"last_name": "Smith"}
  }
}

GET employee/_search
{
    "query": {
        "bool":{
            "must": {
                "match": {"last_name": "Smith"}
            },
            "filter": {
                "range": {
                  "age": {
                    "gt": 30
                  }
                }
            }
        }
    }
}
```

全文搜索

```
GET employee/_search
{
    "query": {
        "match": {
          "about": "rock climbing"
        }
    }
}
```

短语搜索

```
GET employee/_search
{
    "query": {
        "match_phrase": {
          "about": "rock climbing"
        }
    }
}
```

高亮搜索

```
GET employee/_search
{
    "query": {
        "match_phrase": {
          "about": "rock climbing"
        }
    },
    "highlight": {
      "fields": {
        "about": {}
      }
    }
}
```

聚合分析

员工兴趣统计

```
GET employee/_search
{
    "aggs": {
    "all_interests": {
      "terms": { "field": "interests" }
    }
  }
}
```

在 `interests` 上进行聚合会报错。

```
    "root_cause" : [
      {
        "type" : "illegal_argument_exception",
        "reason" : "Text fields are not optimised for operations
      }
    ]
```

正确写法应该是：

```
GET employee/_search
{
    "aggs": {
    "all_interests": {
      "terms": { "field": "interests.keyword" }
    }
  }
}
```

具体分析看这里：ELK - Elasticsearch权威指南里的聚合分析报错: Text fields are not optimised for operations that require per-document field data like aggregations and sorting

某些员工兴趣

可以加组合查询条件：

```
GET employee/_search
{
  "query": {
    "match": {
      "last_name": "smith"
    }
  },
  "aggs": {
    "all_interests": {
      "terms": {
        "field": "interests.keyword"
      }
    }
  }
}
```

兴趣的平均年龄

也可以汇总分析，有某个兴趣爱好的所有员工的平均年龄。

```
GET employee/_search
{
    "aggs" : {
        "all_interests" : {
            "terms" : { "field" : "interests.keyword" },
            "aggs" : {
                "avg_age" : {
                    "avg" : { "field" : "age" }
                }
            }
        }
    }
}
```

相关文章

ELK - Elasticsearch权威指南里的聚合分析报错: Text fields are not optimised for operations that require per-document field data like aggregations and sorting

ELK - Elasticsearch权威指南里的聚合分析报错: Text fields are not optimised for operations that require per-document field data like aggregations and sorting

Elasticsearch权威指南是指官网的《Elasticsearch：权威指南》。

聚合分析

```
GET employee/_search
{
    "aggs": {
    "all_ages": {
      "terms": { "field": "age" }
    }
  }
}

GET employee/_search
{
    "aggs": {
    "all_interests": {
      "terms": { "field": "interests" }
    }
  }
}
```

在 `interests` 上进行聚合会报错，而 `age` 没问题。

```
    "root_cause" : [
      {
        "type" : "illegal_argument_exception",
        "reason" : "Text fields are not optimised for operations
      }
    ]
```

因为 `interests` 的 `type` 是 `text` ，而 `age` 不是，
`text` 或 `annotated_text` 字段 `doc_values` 默认为 `false` 。

简单理解，就是 `text` 字段作为一个整体，默认没有索引。

```
GET employee/_mapping
{
  "employee" : {
    "mappings" : {
      "properties" : {
        "about" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        },
        "age" : {
          "type" : "long"
        },
        "first_name" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        },
        "interests" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        },
        "last_name" : {
          "type" : "text",
          "fields" : {
            "keyword" : {
              "type" : "keyword",
              "ignore_above" : 256
            }
          }
        }
      }
    }
  }
}
```

不过 `text` 分词之后的 `keyword` 是有索引的，因而可以对 `interests.keyword` 进行聚合。

```
GET employee/_search
{
    "aggs": {
    "all_interests": {
      "terms": { "field": "interests.keyword" }
    }
  }
}
```

也可以 `set fielddata=true` ，不过不推荐。

> Fielddata is disabled on text fields by default.
>
> Fielddata can consume a lot of heap space, especially when loading high cardinality text fields. Once fielddata has been loaded into the heap, it remains there for the lifetime of the segment. Also, loading fielddata is an expensive process which can cause users to experience latency hits. This is why fielddata is disabled by default.
>
> ```
> PUT employee/_mapping
> {
>  "properties": {
>  "interests": {
>  "type": "text",
>  "fielddata": true
>  }
>  }
> }
> ```

相关文章

ELK - Elasticsearch权威指南里的简单搜索例子

Filebeat重启之后不读Log了？Registry file is too large

Filebeat重启之后突然不读Log了，看起来又是正常运行。

问一下有没有什么Bug，说 `Registry file is too large` 。

后来我看了一下，只有几十M，不大呀？

回复说，几十M已经太大了。

果然，对大的理解不太一样。

看一下文档，大概明白 `大` 的意思了，因为这其实是状态文件，Filebeat每时每刻都要读写。

> Filebeat keeps the state of each file and persists the state to disk in the registry file. The file state is used to continue file reading at a previous position when Filebeat is restarted. If a large number of new files are produced every day, the registry file might grow to be too large. To reduce the size of the registry file, there are two configuration options available: clean_removed and clean_inactive.
>
> For old files that you no longer touch and are ignored (see ignore_older), we recommended that you use clean_inactive. If old files get removed from disk, then use the clean_removed option.

所以解决办法就是：清理过期文件记录，增加写缓存时间减少IO。

```
vi /etc/filebeat/filebeat.yml

ignore_older: 24h
clean_inactive: 25h

filebeat.registry.flush: 5s
```

ELK - Hearthbeat实现服务监控

Hearthbeat，心跳，顾名思义，Hearthbeat可以用来定时探测服务是否正常运行。

Hearthbeat支持ICMP、TCP 和 HTTP，也支持TLS、身份验证和代理。

Hearthbeat能够与Logstash、Elasticsearch和Kibana无缝协作。安装Hearthbeat，添加需要监控的服务，配置好Elasticsearch和Kibana，即可将结果输出到Elasticsearch，并在Kibana显示出来。

Kibana无须再配置Dashboard，直接点击 `Uptime` 菜单即可看到结果。

Elasticsearch和Kibana的安装可参考：快速搭建ELK（7.2.0）

下载

```
curl -L -O https://artifacts.elastic.co/downloads/beats/heartbea
tar xzvf heartbeat-7.2.0-linux-x86_64.tar.gz -C /opt/elk/
```

配置

```
cd /opt/elk/heartbeat-7.2.0-linux-x86_64/
vi heartbeat.yml
```

比如通过简单地探测console url来监控Websphere是否正常运行。

监控TCP可参考注释里的例子。Schedule表示每10秒监测一次。

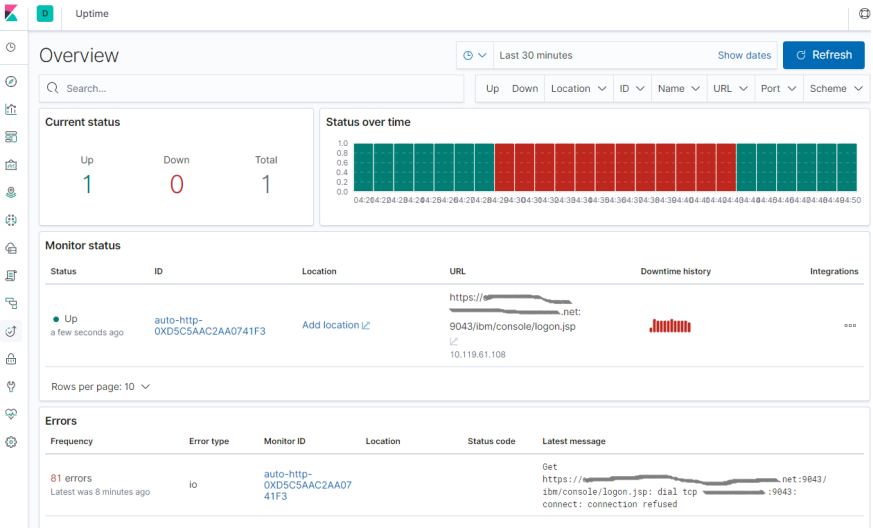
```
# Configure monitors inline
heartbeat.monitors:
- type: http

  # List or urls to query
  urls: ["https://dummy.hostname.net:9043/ibm/console/logon.jsp"

  # Configure task schedule
  schedule: '@every 10s'

#- type: tcp
  #hosts: ["192.168.1.101:1414"]
  #schedule: '@every 10s'


setup.kibana:

  # Kibana Host
  # Scheme and port can be left out and will be set to the defau
  # In case you specify and additional path, the scheme is requi
  # IPv6 addresses should always be defined as: https://[2001:db
  host: "localhost:5601"

output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["localhost:9200"]
```

启动

```
chown elk:elk -R /opt/elk/heartbeat-7.2.0-linux-x86_64/
sudo chmod o+rx -R /opt/elk/heartbeat-7.2.0-linux-x86_64/

sudo -u elk ./heartbeat setup

sudo -u elk nohup ./heartbeat -e &
```

验证

- Open http://localhost:5601/
- Go to Uptime.
- 重启被监控服务


Uptime
Overview
Last 30 minutes
Show dates
Refresh
Search...
Up
Down
Location
ID
Name
URL
Port
Scheme
Current status
Up
1
Down
0
Total
1
Status over time
Monitor status
Status
ID
Location
URL
Downtime history
Integrations
Up
a few seconds ago
auto-http-0XD5C5AAC2AA0741F3
Add location
https:// [redacted] .net: 9043/ibm/console/logon.jsp
10.119.61.108
Rows per page: 10
Errors
Frequency
Error type
Monitor ID
Location
Status code
Latest message
81 errors
Latest was 8 minutes ago
io
auto-http-0XD5C5AAC2AA0741F3
Get https:// [redacted] .net:9043/ibm/console/logon.jsp: dial tcp [redacted] :9043: connect: connection refused

Logstash-keystore保存Elasticsearch或其他密码

方法好简单，文档好齐全，但还是有坑。

```
set +o history

# Have to setup keystore password to encrypt keystore, before ru
export LOGSTASH_KEYSTORE_PASS=changeme

# Have to save the password for service to kick start (installed
echo "LOGSTASH_KEYSTORE_PASS=changeme" > /etc/sysconfig/logstash

set -o history

# Have to add `--path.settings` paramemter, then logstash.keysto
/usr/share/logstash/bin/logstash-keystore --path.settings /etc/l
/usr/share/logstash/bin/logstash-keystore --path.settings /etc/l
/usr/share/logstash/bin/logstash-keystore --path.settings /etc/l
```

Bug: Don't run the command under /etc/logstash, otherwise will see error below.

```
2020-11-16 16:10:37,143 main ERROR Unable to locate appender "${
2020-11-16 16:10:37,144 main ERROR Unable to locate appender "${
2020-11-16 16:10:37,144 main ERROR Unable to locate appender "${
2020-11-16 16:10:37,145 main ERROR Unable to locate appender "${
```

ELK - Logstash收集多种Filebeat数据，如何区分不同来源

比如Filebeat。

Logstash Filer Grok

Logstash配置可以使用 `if [fields][type]` 来区分来源。

test.conf

```
input {
  beats {
    port => 5044
  }
}

filter {
  if [fields][type] == "test" {
    grok {
      match => { "message" => "%{DATESTAMP:logTime}%-%{GREEDYDAT
      overwrite => [ "message" ]
    }
  }
}

output {
  if [fields][type] == "test" {
    stdout { codec => rubydebug }
  }
  if "_grokparsefailure" in [tags] {
    file { "path" => "/tmp/grok_failures.log" }
  }
}
```

Filebeat

Filebeat配置相应的加上 `fields.type` 定义。

```
vi /etc/filebeat/filebeat.yml
```

```
- type: log

  # Change to true to enable this input configuration.
  enabled: true

  # Paths that should be crawled and fetched. Glob based paths.
  paths:
    - /var/log/*.log
  fields:
    type: test

#output.elasticsearch:
  # Array of hosts to connect to.
  #hosts: ["localhost:9200"]

output.logstash:
  # The Logstash hosts
  hosts: ["remotehost:5044"]
```

Logstash配置调试技巧：Filter Grok日志模式Pattern匹配之类的确实很麻烦

之前我也很不耐烦，不过掌握了以下一些小技巧之后，感觉好多了。

Logstash Config Test and Exit

```
/usr/share/logstash/bin/logstash --config.test_and_exit --path.s
```

Logstash Config Automatic Reload

```
/usr/share/logstash/bin/logstash --path.settings /etc/logstash -
```

Logstash Grok Debugger

一般用网页，因为不用登陆。

- Kibana -> Dev Tools -> Grok Debugger
- http://grokdebug.herokuapp.com/

Logstash Input

接受命令行输入。

```
input {
  stdin {}
}
```

不过用了 `stdin` 之后不支持 `--config.reload.automatic` ，所以实际上我还是用Filebeat多。

```
input {
  beats {
    port => 5044
  }
}
```

只要简单append文本到目标文件即可。

Force Filebeat to Send Data Again

也可以让Filebeat重新再发送一次所有文件。

```
rm -rf /var/lib/filebeat/registry/filebeat
systemctl restart filebeat
```

Logstash Output

输出到控制台，输出不匹配的到文件。

```
output {
  stdout { codec => rubydebug }
  if "_grokparsefailure" in [tags] {
    file { "path" => "/tmp/grok_failures.log" }
  }
}
```

P.S.

一开始感觉挺复杂，后来发现其实很多日志都不规范，而且一般也不需要全匹配，简单使用就足够了。

常用DATESTAMP，DATA，GREEDYDATA，SPACE等。
特别是SPACE，一开始没想到要用，后来发现总是匹配不好，加上就正常了。

`/usr/share/logstash/vendor/bundle/jruby/2.5.0/gems/logstash-patterns-core-4.1.2/patterns/` https://github.com/logstash-plugins/logstash-patterns-core/tree/master/patterns

Set up basic security for the Elastic Stack plus secured HTTPS traffic

Test version: 7.15

1. Generate HTTP cert with the same CA for transport SSL

```
cd /usr/share/elasticsearch
./bin/elasticsearch-certutil http
```

```
Generate a CSR? [y/N]n
Use an existing CA? [y/N]y
CA Path: /usr/share/elasticsearch/elastic-stack-ca.p12
For how long should your certificate be valid? [5y]
Generate a certificate per node? [y/N]y
node #1 name: appserver01

hostnames:
appserver01

IPs:
192.168.0.111
```

2. unzip elasticsearch-ssl-http.zip -d ssl

```
cp ssl/elasticsearch/appserver01/http.p12 /etc/elasticsearch/
chmod 644 /etc/elasticsearch/http.p12
```

3. vi /etc/elasticsearch/elasticsearch.yml

```
xpack.security.http.ssl.enabled: true
xpack.security.http.ssl.keystore.path: http.p12
```

4. Store the password in the Elasticsearch keystore

```
./bin/elasticsearch-keystore add xpack.security.http.ssl.keystor
```

5. systemctl restart elasticsearch.service

6. Encrypt traffic between Kibana and Elasticsearch

```
cp ssl/kibana/elasticsearch-ca.pem /etc/kibana/

vi /etc/kibana/kibana.yml
```

```
elasticsearch.hosts: ["https://appserver01:9200"]
elasticsearch.ssl.certificateAuthorities: /etc/kibana/elasticsea
```

```
systemctl restart kibana
```

7. Beats to Elasticsearch

e.g. Heartbeat

```
cp ssl/kibana/elasticsearch-ca.pem /etc/heartbeat/

vi /etc/heartbeat/heartbeat.yml

output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["https://appserver01:9200"]
  ssl.certificate_authorities: ["/etc/heartbeat/elasticsearch-ca

systemctl restart heartbeat-elastic.service
```

8. Logstash to Elasticsearch

```
cp ssl/kibana/elasticsearch-ca.pem /etc/logstash/
chmod 644 /etc/logstash/elasticsearch-ca.pem

vi /etc/logstash/conf.d/app01.conf
```

```
output {
  elasticsearch {
    hosts => ["https://appserver01:9200"]
    ssl => true
    ssl_certificate_verification => true
    cacert => ["/etc/logstash/elasticsearch-ca.pem"]
    index => "%{[@metadata][beat]}-%{+YYYY.MM.dd}"
    user => "elastic"
    password => "${ES_PWD}"
  }
}
```

Set up basic security for the Elastic Stack

1. Genrate CA: elastic-stack-ca.p12

cd /usr/share/elasticsearch ./bin/elasticsearch-certutil ca

Input password

2. Generate Cert: elastic-certificates.p12

./bin/elasticsearch-certutil cert --ca elastic-stack-ca.p12

Input password

Complete the following steps for each node in your cluster.

3. cp elastic-certificates.p12 /etc/elasticsearch/

chmod 644 elastic-certificates.p12

4. vi /etc/elasticsearch/elasticsearch.yml

//#cluster.name: my-cluster //#node.name: node-1

xpack.security.transport.ssl.enabled: true
xpack.security.transport.ssl.verification_mode: certificate
xpack.security.transport.ssl.client_authentication: required
xpack.security.transport.ssl.keystore.path: elastic-certificates.p12
xpack.security.transport.ssl.truststore.path: elastic-certificates.p12

5. Store the password in the Elasticsearch keystore

```
./bin/elasticsearch-keystore add
xpack.security.transport.ssl.keystore.secure_password
./bin/elasticsearch-keystore add
xpack.security.transport.ssl.truststore.secure_password
```

6. Restart elasticsearch

ELK - Watcher发送告警邮件和调用接口

相比使用ElastAlert发送告警邮件，ELK提供的Wathcer要简单得多，也可以在发生警报的时候调用Web Service接口。

Configure SMTP

https://www.elastic.co/guide/en/elasticsearch/reference/current/actions-email.html

以上文档提供了多种Email系统的配置方法(elasticsearch.yml)，包括Gmail, Outlook, Microsoft Exchange, Amazon SES。

比如Gmail：

```
xpack.notification.email.account:
    gmail_account:
        profile: gmail
        smtp:
            auth: true
            starttls.enable: true
            host: smtp.gmail.com
            port: 587
            user: <username>
```

当然还要在 `elasticsearch-keystore` 里配置相应的 `password`

```
bin/elasticsearch-keystore add xpack.notification.email.account.
```

实际上公司一般有内部SMTP，只需授权，而无需用户名和密码。

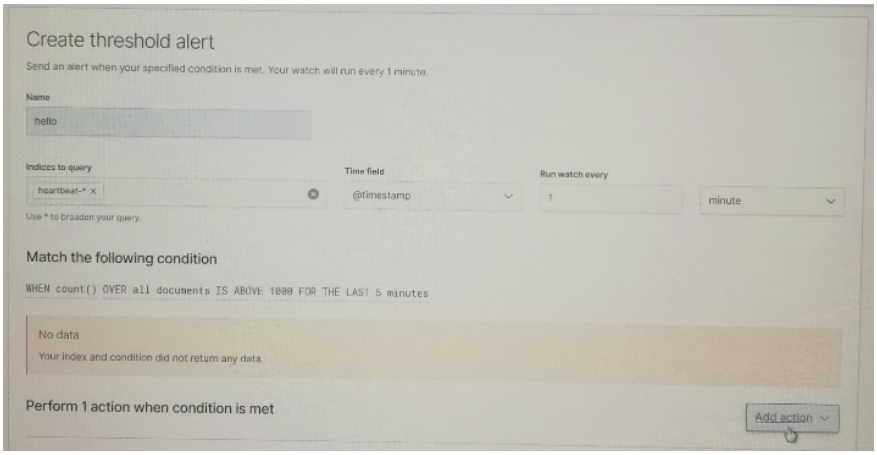
```
xpack.notification.email.account:
  work:
    profile: standard
    email_defaults:
      from: my@dummy.email
    smtp:
      auth: false
      starttls.enable: false
      host: my.dummy.smtp.host
      port: 25
```

Create Watcher

Create Threshold Alert

`Management` -> `Elasticsearch` -> `Watcher` -> `Create threshold alert` .

填入 `Name` ， `Indices` ， `Time field` ， 则会出现 `Add action` 按钮。

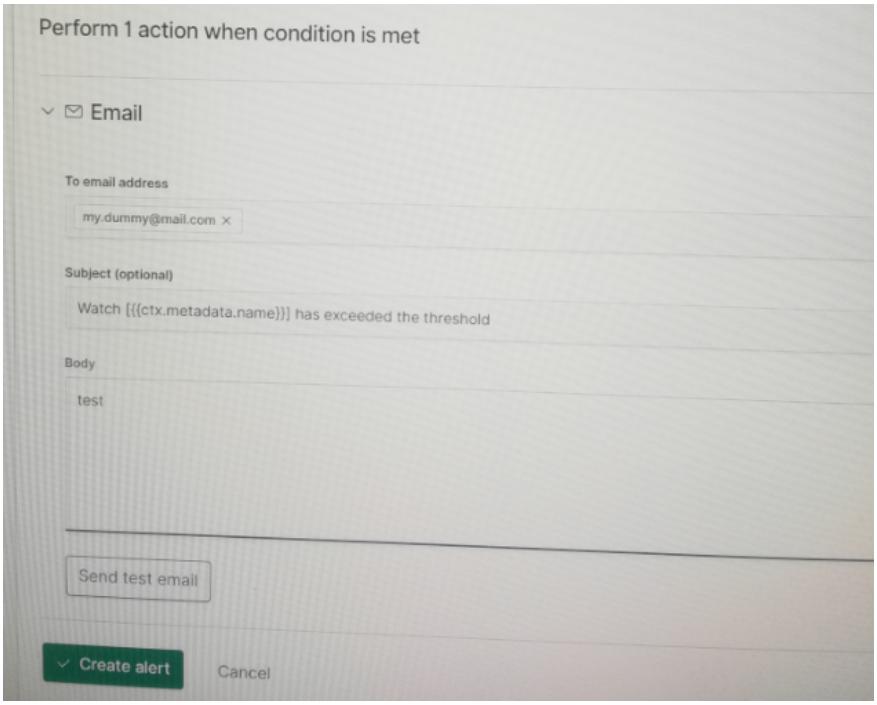


> Watcher supports the following types of actions: email, webhook, index, logging, slack, and pagerduty.

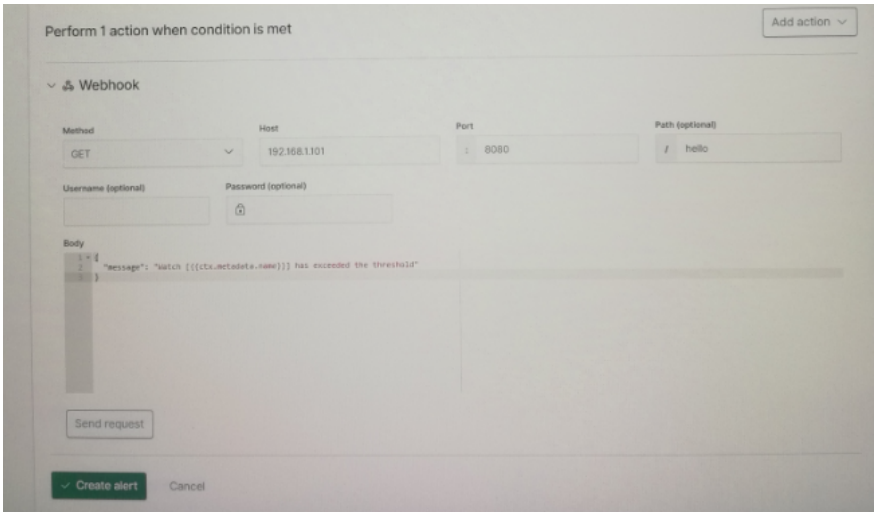
Threshold Alert的主要作用，是它提供了界面，可以简单测试下配置有没有起效果。

比如Email， 填入邮件地址和内容，点击 `Send test email` 。

如果SMTP配置没问题的话，应该可以成功收到邮件。



调用接口则选 `Webhook` ，一样可以直接 `Send request` 进行测试。



注意到7.3还不支持HTTPS，7.6以后才有此选项。 `Advance Watch` 则没有这个问题。

Create Advance Watch

以下是缺省的模板，30分钟执行一次，查询所有indices，因而一般都能执行。

把时间调小，很快就可以在 `elasticsearch.log` 里看到输出的 `text` 。

```
{
  "trigger": {
    "schedule": {
      "interval": "30m"
    }
  },
  "input": {
    "search": {
      "request": {
        "body": {
          "size": 0,
          "query": {
            "match_all": {}
          }
        },
        "indices": [
          "*"
        ]
      }
    }
  },
  "condition": {
    "compare": {
      "ctx.payload.hits.total": {
        "gte": 10
      }
    }
  },
  "actions": {
    "my-logging-action": {
      "logging": {
        "text": "There are {{ctx.payload.hits.total}} documents
      }
    }
  }
}
```

Email Action

发送告警邮件的配置一般长这样。

```
"actions": {
  "send_email": {
    "email": {
      "profile": "standard",
      "to": [
        "my.support@dummy.email"
      ],
      "subject": "ELK Alert - XXX is Down",
      "body": {
        "html": "{{ctx.payload.hits.total}} XXX is over limit,
      }
    }
  }
}
```

Webhook Action

告警时调用接口配置一般长这样。

可以同时支持多个Action。

```
"actions": {
  "my-logging-action": {
    "logging": {
      "level": "info",
      "text": "There are {{ctx.payload.hits.total}} documents
    }
  },
  "my-webhook-action": {
    "webhook": {
      "scheme": "https",
      "host": "my.api.dummy.host",
      "port": 8443,
      "method": "put",
      "path": "api/alert",
      "params": {},
      "headers": {
        "Content-type": "application/json"
      },
      "body": "{status: 1}"
    }
  }
}
```

ELK - X-Pack设置用户密码

之前记录过怎样使用Nginx代理为Kibana添加登录验证功能，但其实Elastic本身也提供了基本的访问控制功能。
虽然相关功能在X-Pack里，但还是可以免费使用的。
参考官网subscriptions，可以看到在 `Basic License` 下的 `Security` 项目中包含了 `Role-based access control` 。

Enable X-Pack Security

Elasticsearch设置密码命令如下。

```
$ sudo /usr/share/elasticsearch/bin/elasticsearch-setup-password

Unexpected response code [500] from calling GET http://192.168.1
It doesn't look like the X-Pack security feature is enabled on t
Please check if you have enabled X-Pack security in your elastic

ERROR: X-Pack Security is disabled by configuration.
```

设置前，需要先enable X-Pack security。

```
sudo vi /etc/elasticsearch/elasticsearch.yml
```

```
xpack.security.enabled: true
xpack.security.transport.ssl.enabled: true
```

SSL也要同时enable，否则报错。

```
[2019-10-10T15:11:09,630][ERROR][o.e.b.Bootstrap          ] [app
[1] bootstrap checks failed
[1]: Transport SSL must be enabled if security is enabled on a [
```

现在可以开始设置密码。
可以看到Elasticsearch预置了许多角色和用户。

```
$ sudo /usr/share/elasticsearch/bin/elasticsearch-setup-password
Initiating the setup of passwords for reserved users elastic,apm
You will be prompted to enter passwords as the process progresse
Please confirm that you would like to continue [y/N]y


Enter password for [elastic]:
Reenter password for [elastic]:
Enter password for [apm_system]:
Reenter password for [apm_system]:
Enter password for [kibana]:
Reenter password for [kibana]:
Enter password for [logstash_system]:
Reenter password for [logstash_system]:
Enter password for [beats_system]:
Reenter password for [beats_system]:
Enter password for [remote_monitoring_user]:
Reenter password for [remote_monitoring_user]:
Changed password for user [apm_system]
Changed password for user [kibana]
Changed password for user [logstash_system]
Changed password for user [beats_system]
Changed password for user [remote_monitoring_user]
Changed password for user [elastic]
```

重启

```
sudo systemctl restart elasticsearch.service
```

需要加密码才能访问。

```
$ curl localhost:9200
{"error":{"root_cause":[{"type":"security_exception","reason":"m

$ curl localhost:9200 -uelastic:<password>
{
  "name" : "appserver01",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "wCDlDy6UTriUSuZuDrqWrw",
  "version" : {
    "number" : "7.3.2",
    "build_flavor" : "default",
    "build_type" : "rpm",
    "build_hash" : "1c1faf1",
    "build_date" : "2019-09-06T14:40:30.409026Z",
    "build_snapshot" : false,
    "lucene_version" : "8.1.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Update Kibana Configure

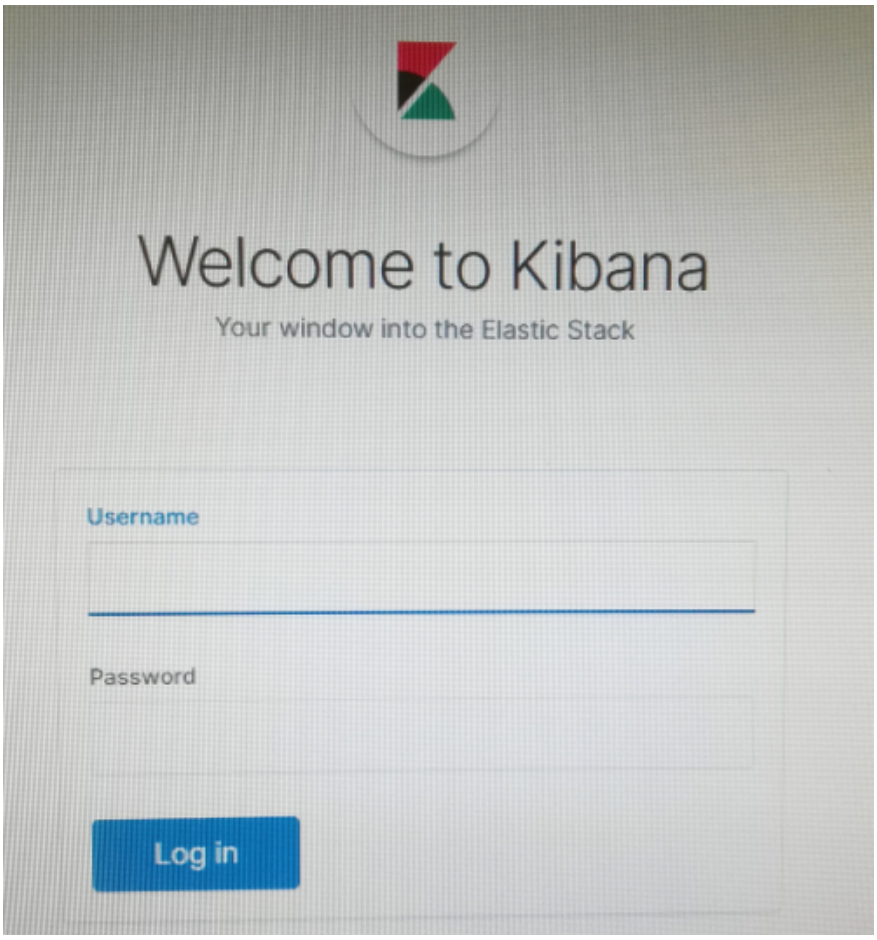
After password setup in Elasticsearch, have to update configure for Kibana, Logstash and Beats. e.g. Kibana

```
sudo vi /etc/kibana/kibana.yml
# sudo vi /etc/heartbeat/heartbeat.yml
```

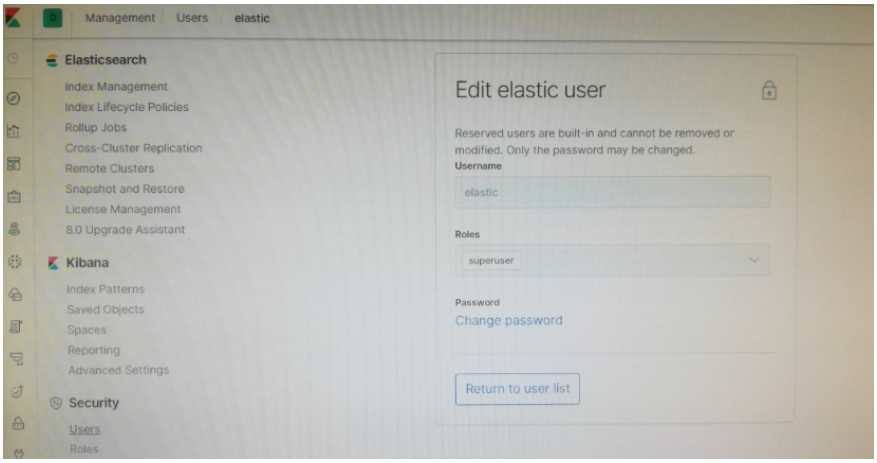
```
elasticsearch.username: "elastic"
elasticsearch.password: "<password>"
```

```
sudo systemctl restart kibana.service
```

After restart, refresh Kibana and you will be asked for username and password now.



To change the password, can go to Kibana Management -> Security -> Users.



Or change data in Elasticsearch directly.

```
POST /_security/user/elastic/_password
{
  "password": "123456"
}
```

ELK - 使用ElastAlert发送邮件

简单记录下之前的研究。

https://buildmedia.readthedocs.org/media/pdf/elastalert/latest/elastalert.pdf

Install Python

之前在ElastAlert最大的坑里说过，要用Python3！

Install Pip

如果没有pip，则需要安装。

```
sudo easy_install pip
```

或 Download get-pip.py: https://bootstrap.pypa.io/get-pip.py

```
sudo python get-pip.py
```

Install Elastalert

```
sudo pip install elastalert
```

需要其他包按提示装好。

安装完后有这些命令可以用。

```
ll /usr/local/bin/ela*
/usr/local/bin/elastalert
/usr/local/bin/elastalert-create-index
/usr/local/bin/elastalert-rule-from-kibana
/usr/local/bin/elastalert-test-rule
```

可以这样运行。

```
sudo python3 /usr/local/bin/elastalert-test-rule --config config
sudo python3 /usr/local/bin/elastalert --verbose --config config
sudo python3 /usr/local/bin/elastalert --verbose --rule down_fre
```

或者这样。

```
sudo python3 -m elastalert.elastalert --verbose --config config.
```

`elastalert-test-rule` 的效果与 `elastalert --debug` 相似，验证规则但不真的发送邮件。

结果

先说结果。

使用 `elastalert-test-rule` 或 `elastalert --debug` ，可以看到自己定义的邮件内容。

```
INFO:elastalert:Queried rule Down frequency rule from 2019-11-18
INFO:elastalert:Skipping writing to ES: {'exponent': 0, 'rule_na
INFO:elastalert:Alert for Down frequency rule at 2019-11-18T03:3

INFO:elastalert:Dear Team, mq dev is down @ 2019-11-18T03:35:10.
### Number hits: 54
> Check time: 2019-11-18T03:35:10.755Z
> IP: 192.168.1.88
> Env: mq dev
> Status: down
>>> Error: dial tcp 192.168.1.88:8888: connect: connection refus
```

正常运行时，提示如下。

```
INFO:elastalert:Sleeping for 59.999735 seconds
INFO:elastalert:Sent email to ['my.dummy@email.com']
INFO:elastalert:Ignoring match for silenced rule Down frequency
INFO:elastalert:Ran Down frequency rule from 2019-11-18 10:10 CS
```

Create Index in Elasticsearch

ElastAlert的状态保存，需要在Elasticsearch里create index。

可以直接用示例配置来试。

```
cp config.yaml.example config.yaml
sudo python3 /usr/local/bin/elastalert-create-index --config con
```

Create Rule

创建Alert规则，主要是怎样query，怎么trigger，定义SMTP和信息内容等。

初次测试，要确保query的条件满足，可以触发邮件发送。

Email那里，遇到一个坑：提示alerts sent，然而邮件并没有收到，乱改一通最后加上 `from_addr` 就正常了。

```
# Alert when the rate of events exceeds a threshold

# (Optional)
# Elasticsearch host
es_host: localhost

# (Optional)
# Elasticsearch port
es_port: 9200

# (OptionaL) Connect with SSL to Elasticsearch
#use_ssl: True

# (Optional) basic-auth username and password for Elasticsearch
#es_username: someusername
#es_password: somepassword

# (Required)
# Rule name, must be unique
name: Down frequency rule

# (Required)
# Type of alert.
# the frequency rule type alerts when num_events events occur wi
type: frequency

# (Required)
# Index to search, wildcard supported
index: heartbeat-*

# (Required, frequency specific)
# Alert when this many documents matching the query occur within
num_events: 50

# (Required, frequency specific)
# num_events must occur within this amount of time to trigger an
timeframe:
  hours: 4

# (Required)
# A list of Elasticsearch filters used for find events
# These filters are joined with AND and nested in a filtered que
# For more info: http://www.elasticsearch.org/guide/en/elasticse
filter:
- term:
    monitor.status: "down"

# (Required)
```

```yaml
# The alert is use when a match is found
alert:
- "email"

# (required, email specific)
# a list of email addresses to send alerts to
email:
- "my.dummy@email.com"

from_addr: "rufeng.pan@email.com"
smtp_host: "dummy.smtp.host"
smtp_port: 25

alert_subject: "ELK Alert - {0} Down @ {1}"
alert_subject_args:
- monitor.name
- "@timestamp"

alert_text_type: alert_text_only
alert_text: |
  Dear Team, {} is down @ {}, please take action!
  ### Number hits: {}
  > Check time: {}
  > IP: {}
  > Env: {}
  > Status: {}
  >>> Error: {}

alert_text_args:
- monitor.name
- "@timestamp"
- num_hits
- "@timestamp"
- monitor.ip
- monitor.name
- monitor.status
- error.message
```

快速搭建ELK（7.2.0）

真正用ELK，才发现ELK的文档那么多，眼花缭乱。
也难怪，每一个单独起来也都是挺好的小工具。
有必要稍微梳理一下。

Elastic Stack

- Beats
- Logstash
- Elasticsearch
- Kibana

Elasticsearch是核心搜索引擎，用于数据缓存、搜索和分析。

Beats是轻量级的数据采集器，有很多种，按需取用。

Elastic Beats	To capture
Auditbeat	Audit data
Filebeat	Log files
Functionbeat	Cloud data
Heartbeat	Availability monitoring
Journalbeat	Systemd journals
Metricbeat	Metrics
Packetbeat	Network traffic
Winlogbeat	Windows event logs

Logstash是数据收集管道，数据通过它进行归集，同时可以对经过的数据进行简单处理，如清洗过滤和润色等。

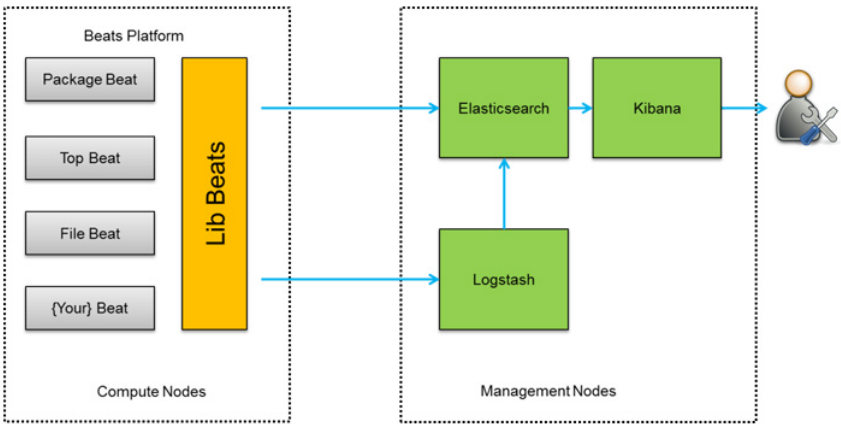
Kibana是ElasticSearch的可视化平台，提供了数据分析和监控的界面和工具集。

https://www.elastic.co/guide/index.html

ELK架构

比较简单实用的架构应该是下面这种，Beats装在要采集数据的终端，将数据发给Logstash，再转给ElasticSearch。

Beats也可以直接发数据给ElasticSearch，但经Logstash可以先进行一些预处理。它们之间也可以进一步引入消息队列。



ELK安装

先不管那么多，把四个软件都下载安装了再说。这里用的是解压版。需要注意的是：要安装相同的版本。

- elasticsearch-7.2.0-linux-x86_64.tar.gz
- filebeat-7.2.0-linux-x86_64.tar.gz
- kibana-7.2.0-linux-x86_64.tar.gz
- logstash-7.2.0.tar.gz

要monitor log files，则装Filebeat。

https://www.elastic.co/guide/en/elastic-stack-get-started/current/get-started-elastic-stack.html

Elasticsearch

直接启动会遇到如下错误。

Can not run elasticsearch as root

不能用root用户运行，转到其他用户即可。当然该用户要有相应权限。

加参数 `-d` 表示在后台运行， `-p` 用来输出PID，这样就可以用 `pkill -F pid` 来关闭程序。

```
chown elkusr:elkusr -R /opt/elk/
cd /opt/elk/elasticsearch-7.2.0/
sudo -u elkusr ./bin/elasticsearch -d -p pid
pkill -F pid
```

成功启动后 `curl` 结果如下。

```
curl localhost:9200
  % Total    % Received % Xferd  Average Speed   Time    Time
                                 Dload  Upload   Total   Spent
100   507  100   507    0     0  31687      0 --:--:-- --:--:--
  "name" : "appserver1",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "DVsCk28AA5WhL2BhCtkxtw",
  "version" : {
    "number" : "7.2.0",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "508c38a",
    "build_date" : "2019-06-20T15:54:18.811730Z",
    "build_snapshot" : false,
    "lucene_version" : "8.0.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

远程访问Elasticsearch

如果ELK组件安装在不同机上，则需要打开远程访问。

- network.host

更新network.host为0.0.0.0

```
vi /opt/elk/elasticsearch-7.2.0/config/elasticsearch.yml
#network.host: 192.168.0.1
network.host: 0.0.0.0
```

- initial_master_nodes and vm.max_map_count

这时启动会遇到如下错误。

```
[2] bootstrap checks failed
[1]: max virtual memory areas vm.max_map_count [65530] is too lo
[2]: the default discovery settings are unsuitable for productio
```

第一个是要提高最大线程数（max_map_count/2）。

```
echo "vm.max_map_count=262144" >> /etc/sysctl.conf
sysctl -p
```

第二个是Elasticsearch7.0以后一些discovery相关默认配置改了，添加如下配置，重启。

```
vi /opt/elk/elasticsearch-7.2.0/config/elasticsearch.yml
cluster.initial_master_nodes: ["node-1"]
```

- 开Linux端口

  ```
  firewall-cmd --zone=public --add-port=9200/tcp --permanent
  firewall-cmd --reload
  ```

- 测试远程访问

用浏览器打开则返回一个JSON。

```
curl appserver1:9200
```

Kibana

远程访问Kibana

Kibana是UI，远程访问肯定要开。

- server.host

```
#server.host: "localhost"
server.host: "0.0.0.0"
```

- 开Linux端口

```
firewall-cmd --add-port=5601/tcp --permanent
firewall-cmd --reload
```

Kibana Logging配置

Kibana日志默认输出到stdout。怕有问题，可以暂时输出到文件。

```
sudo -u elkusr mkdir /opt/elk/kibana-7.2.0-linux-x86_64/logs

vi /opt/elk/kibana-7.2.0-linux-x86_64/config/kibana.yml

#logging.dest = stdout
logging.dest = /opt/elk/kibana-7.2.0-linux-x86_64/logs/
```

Kibana Startup

```
cd /opt/elk/kibana-7.2.0-linux-x86_64
sudo -u elkusr nohup ./bin/kibana &

ps -ef|grep node
tail -f /opt/elk/kibana-7.2.0-linux-x86_64/logs/kibana.log
```

http://appserver1:5601/

Filebeat

以下配置顺便以Kibana的日志来测试，输出到Logstash。

```
vi /opt/elk/filebeat-7.2.0-linux-x86_64/filebeat.yml
filebeat.inputs:
- type: log
  enabled: true
  paths:
    #- /var/log/*.log
    /opt/elk/kibana-7.2.0-linux-x86_64/logs/*.log


output.logstash:
  hosts: ["127.0.0.1:5044"]

cd /opt/elk/filebeat-7.2.0-linux-x86_64
sudo -u elkusr nohup ./filebeat -e &
```

Logstash

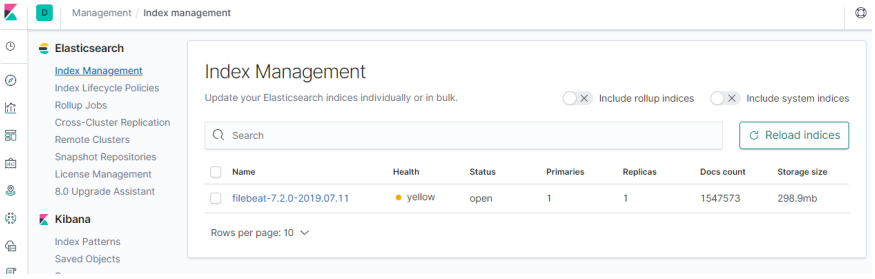
Logstash接收Filebeat输入，然后输出到Elasticsearch。
这里直接用logstash-sample.conf。

```
vi /opt/elk/logstash-7.2.0/config/logstash-sample.conf

# Sample Logstash configuration for creating a simple
# Beats -> Logstash -> Elasticsearch pipeline.

input {
  beats {
    port => 5044
  }
}

output {
  elasticsearch {
    hosts => ["http://localhost:9200"]
    index => "%{[@metadata][beat]}-%{[@metadata][version]}-%{+YY
    #user => "elastic"
    #password => "changeme"
  }
}

cd /opt/elk/logstash-7.2.0/
sudo -u elkusr nohup ./bin/logstash -f ./config/logstash-sample.
```

验证

打开http://appserver1:5601/.

找到Management -> Index Management，可以看到新的index：filebeat-7.2.0-2019.07.11，说明数据已经成功输入Elasticsearch。



快速搭建ELK7.3

A quick start guild of ELK 7.3.2, including package installation and simple test.

好吧，其实是发现之前的快速搭建ELK7.2有些坑，决定还是再来一次。

安装版相比直接解压版还是有些好处的，最起码安装完就是Service，省却打命令行的许多麻烦。

Download and install

- elasticsearch-7.3.2-x86_64.rpm
- kibana-7.3.2-x86_64.rpm
- logstash-7.3.2.rpm
- filebeat-7.3.2-x86_64.rpm
- heartbeat-7.3.2-x86_64.rpm

https://www.elastic.co/guide/en/elastic-stack-get-started/current/get-started-elastic-stack.html

Elasticsearch

Install

```
$ sudo rpm -ivh elasticsearch-7.3.2-x86_64.rpm
warning: elasticsearch-7.3.2-x86_64.rpm: Header V4 RSA/SHA512 Si
Preparing...                          ##########################
Creating elasticsearch group... OK
Creating elasticsearch user... OK
Updating / installing...
   1:elasticsearch-0:7.3.2-1          ##########################
### NOT starting on installation, please execute the following s
 sudo systemctl daemon-reload
 sudo systemctl enable elasticsearch.service
### You can start elasticsearch service by executing
 sudo systemctl start elasticsearch.service
Created elasticsearch keystore in /etc/elasticsearch
```

Startup

```
sudo systemctl daemon-reload
sudo systemctl enable elasticsearch.service

sudo systemctl status elasticsearch.service
sudo systemctl start elasticsearch.service
### Check process
# ps -ef|grep ela
### Check installed files location
# sudo rpm -ql elasticsearch
```

Test

```
$ curl localhost:9200
{
  "name" : "appserver01",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "wCDlDy6UTriUSuZuDrqWrw",
  "version" : {
    "number" : "7.3.2",
    "build_flavor" : "default",
    "build_type" : "rpm",
    "build_hash" : "1c1faf1",
    "build_date" : "2019-09-06T14:40:30.409026Z",
    "build_snapshot" : false,
    "lucene_version" : "8.1.0",
    "minimum_wire_compatibility_version" : "6.8.0",
    "minimum_index_compatibility_version" : "6.0.0-beta1"
  },
  "tagline" : "You Know, for Search"
}
```

Startup Commands

All other installations are similar, you can find the services list as below.

```
sudo systemctl start elasticsearch.service
sudo systemctl start kibana.service
sudo systemctl start logstash.service
sudo systemctl start filebeat.service
sudo systemctl start heartbeat-elastic.service
```

Remote Access to Elasticsearch

```
sudo vi /etc/elasticsearch/elasticsearch.yml
```

```
#network.host: 192.168.0.1
network.host: 0.0.0.0

cluster.initial_master_nodes: node-1
```

The last configure change is for bootstrap error below.

```
sudo tail  /var/log/elasticsearch/elasticsearch.log
```

```
[2019-10-10T10:06:24,340][ERROR][o.e.b.Bootstrap          ] [app
[1] bootstrap checks failed
[1]: the default discovery settings are unsuitable for productio
```

没有遇到7.2解压版里的 `vm.max_map_count is too low` 的问题。

粗略看一下，它是放到启动的script里了。

```
sudo view /etc/init.d/elasticsearch

MAX_OPEN_FILES=65535
MAX_MAP_COUNT=262144

sysctl -q -w vm.max_map_count=$MAX_MAP_COUNT
```

Remote Access to Kibana

```
# curl localhost:5601
# curl localhost:5601/app/kibana
sudo vi /etc/kibana/kibana.yml
```

```
#server.host: "localhost"
server.host: "0.0.0.0"
```

```
sudo systemctl restart kibana.service
```

Now Kibana can be accessed from another host browser
http://192.168.1.88:5601/

安装路径问题

开始也尝试装到指定目录，但发现装完后Service里的路径并没有改过来，说明还需要些手动工作，还是不折腾了。

```
$ sudo rpm -ivh --prefix=/opt/elk elasticsearch-7.3.2-x86_64.rpm
warning: elasticsearch-7.3.2-x86_64.rpm: Header V4 RSA/SHA512 Si
Preparing...                          ##########################
Creating elasticsearch group... OK
Creating elasticsearch user... OK
Updating / installing...
   1:elasticsearch-0:7.3.2-1          ##########################
### NOT starting on installation, please execute the following s
 sudo systemctl daemon-reload
 sudo systemctl enable elasticsearch.service
### You can start elasticsearch service by executing
 sudo systemctl start elasticsearch.service
/var/tmp/rpm-tmp.JyvHeM: line 8: /usr/share/elasticsearch/bin/el
chown: cannot access ‘/etc/elasticsearch/elasticsearch.keystore’
chmod: cannot access ‘/etc/elasticsearch/elasticsearch.keystore’
md5sum: /etc/elasticsearch/elasticsearch.keystore: No such file
warning: %posttrans(elasticsearch-0:7.3.2-1.x86_64) scriptlet fa
```

重新安装问题

因为机器里原本有旧版本的ELK，重新安装Elasticsearch后发现启动不了。

网上说已经有一个在运行，但我这里并没有发现。

最后再一次重装解决了，重装之前把相关路径下的所有文件都删了一遍。

failed to obtain node locks

Failed to startup Elasticsearch with below error.

```
[2019-09-23T16:41:39,751][ERROR][o.e.b.Bootstrap          ] [app
java.lang.IllegalStateException: failed to obtain node locks, tr
```

Not work even after nodes configure is updated as below.

```
node.max_local_storage_nodes: 2
```

The issue was solved after removing all related folders, and we also added CPU and memory (reboot) during the period.

Maybe the files were not cleaned up in the related folders when uninstalled old version.

```
$ sudo rpm -ql elasticsearch>elasticsearch_pkg.log
$ sudo rpm -ev elasticsearch
Preparing packages...
Stopping elasticsearch service... OK
elasticsearch-0:5.4.0-1.noarch
Deleting log directory... OK
$ sudo rpm -ql elasticsearch
package elasticsearch is not installed
```

```
find / -iname "elasticsearch"
```

Remove all old version files found.

```
sudo rm -rf /usr/share/elasticsearch
sudo rm -rf /etc/elasticsearch
sudo rm -rf /var/log/elasticsearch
sudo rm -rf /var/lib/elasticsearch
```

Apache Log4j2 Remote Code Execution (RCE) Vulnerability - CVE-2021-44228 - ESA-2021-31

还是稍微记录一下这个害得大家加班的大bug。

发生版本

Log4j versions 2.0 through 2.14.1。

（更新到2.15也不行。）

解决方案

升级到2.17.0

2.16还是有bug。

https://logging.apache.org/log4j/2.x/

如果不能升级

1. 加JVM参数：-Dlog4j2.formatMsgNoLookups=true
2. 如果没有用，直接删掉：zip -q -d log4j-core-*.jar org/apache/logging/log4j/core/lookup/JndiLookup.class

后话

一顿操作猛如虎，感觉还是有些反应过度了。

一般都是直接扫描，一旦发现，换！

但是，不是所有的系统都是面向网络的，一个后台小工具，急什么。

要实现这个攻击，一般要先构造一个HTTP请求，然后把payload放到Header里，比如：User-Agent。

没有暴露到网络的应用，原则上来说是没有很大风险的。

https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44228

https://blogs.juniper.net/en-us/security/apache-log4j-vulnerability-cve-2021-44228-raises-widespread-concerns

Logback SMTPAppender Send Mail When Log Error

> The SMTPAppender relies on the JavaMail API. It has been tested with JavaMail API version 1.4. The JavaMail API requires the JavaBeans Activation Framework package.

```
<configuration>
  <appender name="EMAIL" class="ch.qos.logback.classic.net.SMTPA
    <evaluator class="ch.qos.logback.classic.boolex.OnErrorEvalu
    <smtpHost>${smtpHost}</smtpHost>
    <to>${to}</to>
    <from>${from}</from>
    <subject>%logger{20} - %m</subject>
    <layout class="ch.qos.logback.classic.html.HTMLLayout"/>

    <cyclicBufferTracker class="ch.qos.logback.core.spi.CyclicBu
      <!-- send just one log entry per email -->
      <bufferSize>1</bufferSize>
    </cyclicBufferTracker>
  </appender>

  <root level="DEBUG">
    <appender-ref ref="EMAIL" />
  </root>
</configuration>
```

OnErrorEvaluator

默認ERROR以上才send email，重寫即可改變觸發條件，比如遇到指定Exception。

> If the Evaluator property is not set, the SMTPAppender defaults to an OnErrorEvaluator instance which triggers email transmission when it encounters an event of level ERROR. While triggering an outgoing email in response to an error is relatively reasonable, it is possible to override this default behavior by providing a different implementation of the EventEvaluator interface.

```
public class OnErrorEvaluator extends EventEvaluatorBase<ILoggin

    /**
     * Return true if event passed as parameter has level ERROR
     * false otherwise.
     */
    public boolean evaluate(ILoggingEvent event) throws NullPoin
        return event.getLevel().levelInt >= Level.ERROR_INT;
    }
}
```

OnMarkerEvaluator

根據Logger參數決定是否send mail。

```
Marker notifyAdmin = MarkerFactory.getMarker("NOTIFY_ADMIN");
logger.error(notifyAdmin,
  "This is a serious an error requiring the admin's attention",
   new Exception("Just testing"));
```

```
<configuration>
  <appender name="EMAIL" class="ch.qos.logback.classic.net.SMTPA
    <evaluator class="ch.qos.logback.classic.boolex.OnMarkerEval
      <marker>NOTIFY_ADMIN</marker>
      <!-- you specify add as many markers as you want -->
      <marker>TRANSACTION_FAILURE</marker>
    </evaluator>
    <smtpHost>${smtpHost}</smtpHost>
    <to>${to}</to>
    <from>${from}</from>
    <layout class="ch.qos.logback.classic.html.HTMLLayout"/>
  </appender>

  <root>
    <level value ="debug"/>
    <appender-ref ref="EMAIL" />
  </root>
</configuration>
```

Other Appenders

http://logback.qos.ch/manual/appenders.html

- FileAppender, RollingFileAppender
- SocketAppender, SSLSocketAppender
- ServerSocketAppender, SSLServerSocketAppender
- DBAppender
- SyslogAppender
- SiftingAppender

Spring - Schedule Task 如何动态改写Cron配置

使用Spring @Scheduled标签可以很简单地定义Scheduled Task，但是有时我们需要在程序里动态地改写Cron的配置。

什么时候呢？

额，比如：

老板觉得Cron配置太难看了，想直接这样： `10: 15`

Scheduling Tasks的常规使用

两个标签： `@EnableScheduling` ， `@Scheduled`

```
@SpringBootApplication
@EnableScheduling
public class SchedulingTasksApplication {

    public static void main(String[] args) {
        SpringApplication.run(SchedulingTasksApplication.class);
    }
}
```

```
public class ScheduleTaskSimpleJob {

    @Scheduled(cron = "0 15 10 * * ?")
    public void scheduleCronTask() {

        long now = System.currentTimeMillis() / 1000;
        System.out.println(
        "schedule tasks using cron jobs - " + now);
    }
}
```

动态改写Cron

Implements `SchedulingConfigurer` 就可以，想怎么改怎么改。

```
public class ScheduleTaskSimpleJob implements SchedulingConfigur

    public void scheduleCronTask() {

        long now = System.currentTimeMillis() / 1000;
        System.out.println(
        "schedule tasks using cron jobs - " + now);
    }

    @Override
    public void configureTasks(ScheduledTaskRegistrar taskRegist
        taskRegistrar.addTriggerTask(new Runnable() {
            @Override
            public void run() {
                scheduleCronTask();
            }
        }, new Trigger() {
            @Override
            public Date nextExecutionTime(TriggerContext trigger
                //TODO 将时间配置10：15转换为cron
                String cron = "0 15 10 * * ?";
                CronTrigger trigger = new CronTrigger(cron);

                Date nextExecDate = trigger.nextExecutionTime(tr
                return nextExecDate;
            }
        });

    }
}
```

发现ThreadPoolExecuter的行为模式跟自己之前的理解有些出入（坑），内在逻辑并不是表面上看那么直观，特地记录一下。

ThreadPoolExecutor构造函数

```java
/**
 * Creates a new {@code ThreadPoolExecutor} with the given i
 * parameters.
 *
 * @param corePoolSize the number of threads to keep in the
 *        if they are idle, unless {@code allowCoreThreadTim
 * @param maximumPoolSize the maximum number of threads to a
 *        pool
 * @param keepAliveTime when the number of threads is greate
 *        the core, this is the maximum time that excess idl
 *        will wait for new tasks before terminating.
 * @param unit the time unit for the {@code keepAliveTime} a
 * @param workQueue the queue to use for holding tasks befor
 *        executed.  This queue will hold only the {@code Ru
 *        tasks submitted by the {@code execute} method.
 * @param threadFactory the factory to use when the executor
 *        creates a new thread
 * @param handler the handler to use when execution is block
 *        because the thread bounds and queue capacities are
 * @throws IllegalArgumentException if one of the following
 *         {@code corePoolSize < 0}<br>
 *         {@code keepAliveTime < 0}<br>
 *         {@code maximumPoolSize <= 0}<br>
 *         {@code maximumPoolSize < corePoolSize}
 * @throws NullPointerException if {@code workQueue}
 *         or {@code threadFactory} or {@code handler} is nu
 */
public ThreadPoolExecutor(int corePoolSize,
                          int maximumPoolSize,
                          long keepAliveTime,
                          TimeUnit unit,
                          BlockingQueue<Runnable> workQueue,
                          ThreadFactory threadFactory,
                          RejectedExecutionHandler handler)
    if (corePoolSize < 0 ||
        maximumPoolSize <= 0 ||
        maximumPoolSize < corePoolSize ||
        keepAliveTime < 0)
        throw new IllegalArgumentException();
    if (workQueue == null || threadFactory == null || handle
        throw new NullPointerException();
    this.acc = System.getSecurityManager() == null ?
            null :
            AccessController.getContext();
    this.corePoolSize = corePoolSize;
    this.maximumPoolSize = maximumPoolSize;
    this.workQueue = workQueue;
    this.keepAliveTime = unit.toNanos(keepAliveTime);
```

```
        this.threadFactory = threadFactory;
        this.handler = handler;
    }
```

也可使用缺省的 `ThreadFactory` 或 `RejectedExecutionHandler` ，通过其他构造函数，如：

```
    public ThreadPoolExecutor(int corePoolSize,
                              int maximumPoolSize,
                              long keepAliveTime,
                              TimeUnit unit,
                              BlockingQueue<Runnable> workQueue)
        this(corePoolSize, maximumPoolSize, keepAliveTime, unit,
             Executors.defaultThreadFactory(), defaultHandler);
    }
```

ThreadPoolExecutor参数说明

- corePoolSize

线程池核心线程数，即使处于闲置状态也不会回收，除非将 `allowCoreThreadTimeOut` 设置为 `true` 。

- maximumPoolSize

线程池所能容纳的最大线程数。

- keepAliveTime

当线程数超过核心线程数时，这些非核心线程闲置超过 `keepAliveTime` 就会被回收。

- unit

`keepAliveTime` 的时间单位，如 `TimeUnit.SECONDS` 。

- workQueue

工作队列，保存通过 `execute` 方法提交的还没有执行的 `Runnable` 任务。

- threadFactory

创建线程的工厂类，可以按需重写，默认的工厂类如下：

```
    /**
     * The default thread factory
     */
    static class DefaultThreadFactory implements ThreadFactory {
        private static final AtomicInteger poolNumber = new Atom
        private final ThreadGroup group;
        private final AtomicInteger threadNumber = new AtomicInt
        private final String namePrefix;

        DefaultThreadFactory() {
            SecurityManager s = System.getSecurityManager();
            group = (s != null) ? s.getThreadGroup() :
                                  Thread.currentThread().getThre
            namePrefix = "pool-" +
                          poolNumber.getAndIncrement() +
                         "-thread-";
        }

        public Thread newThread(Runnable r) {
            Thread t = new Thread(group, r,
                                  namePrefix + threadNumber.getA
                                  0);
            if (t.isDaemon())
                t.setDaemon(false);
            if (t.getPriority() != Thread.NORM_PRIORITY)
                t.setPriority(Thread.NORM_PRIORITY);
            return t;
        }
    }
```

- handler

`RejectedExecutionHandler` 是用来定义线程资源耗尽时的拒绝处理策略，缺省是 `AbortPolicy` ，抛出拒绝处理异常。 线程池的使用应该要能满足业务的需求，从这个意义上来说，抛出异常是合适的。遇到此异常应该首先回去检视线程池的参数定义而非改变策略。

```
    /**
     * A handler for rejected tasks that throws a
     * {@code RejectedExecutionException}.
     */
    public static class AbortPolicy implements RejectedExecution
        /**
         * Creates an {@code AbortPolicy}.
         */
        public AbortPolicy() { }

        /**
         * Always throws RejectedExecutionException.
         *
         * @param r the runnable task requested to be executed
         * @param e the executor attempting to execute this task
         * @throws RejectedExecutionException always
         */
        public void rejectedExecution(Runnable r, ThreadPoolExec
            throw new RejectedExecutionException("Task " + r.toS
                                                 " rejected from
                                                 e.toString());
        }
    }
```

ThreadPoolExecutor行为模式

- 1. `线程数 <= corePoolSize` ，新建一个核心线程执行任务
- 2. `corePoolSize < 线程数 <= maximumPoolSize` ，将任务移入队列等待
- 3.如果队列已满，则新建非核心线程执行任务
- 4.队列已满，且 `线程数 > maximumPoolSize` ，则由 `RejectedExecutionHandler` 抛出异常

所以：
`线程池的最大运行线程数 = 核心线程数 + 非核心线程数`
`线程池的任务吞吐量 = 最大运行线程数 + 任务队列容量` 。

ThreadPoolExecutor的行为模式与所使用的BlockingQueue有很大关系。当使用SynchronousQueue时，任务队列容量为0，如果 `线程数 > corePoolSize` ，直接新建非核心线程执行后续任务，直到超过 `maximumPoolSize` 抛出异常。
当使用没有大小限制的 `LinkedBlockingDeque` 时，任务队列容量无限， `maximumPoolSize` 限制失去意义，任何时候最多只有 `corePoolSize` 个线程正在运行，新任务可以持续添加到任务队列中等待。

ThreadPoolExecutor验证实例

```
package pan.rufeng.demo;

import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.SynchronousQueue;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

public class DemoThreadPoolExecuter {
    public static void main(String[] args) throws InterruptedExc
        System.out.println("Test begin");
        Runnable myRunnable = new Runnable() {
            @Override
            public void run() {
                try {
                    Thread.sleep(2000);
//                    System.out.println(Thread.currentThread().
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }

            }
        };

        // Test case 1: SynchronousQueue, 6 tasks, corePoolSize
        // keepAliveTime to 6
        // maxmumPoolSize to 5
        // Test case 2: LinkedBlockingQueue without size limit,
        // Test case 3: ArrayBlockingQueue或LinkedBlockingQueue
        ThreadPoolExecutor tpe = new ThreadPoolExecutor(3, 6, 5,
        executeRunnable3Times("第一次提交: ", myRunnable, tpe);
        executeRunnable3Times("第二次提交: ",myRunnable, tpe);
        Thread.sleep(8000);
        System.out.println("   8秒之后: corePoolSize: " + tpe.get

    }

    private static void executeRunnable3Times(String title, Runn
        tpe.execute(myRunnable);
        tpe.execute(myRunnable);
        tpe.execute(myRunnable);
        System.out.println(title + "corePoolSize: " + tpe.getCor
    }
}
```

总共提交6个任务，每个任务睡眠2秒，分两次提交，每次提交3个任务，提交之后打印Executor状况，主线程等待8秒之后再一次打印Executor最终状态（验证线程回收）。

- Test case 1: SynchronousQueue, 6 tasks, corePoolSize 4, maxmumPoolSize 6, keepAliveTime 5

```
ThreadPoolExecutor tpe = new ThreadPoolExecutor(4, 6, 5, Tim
```

结果：第二次提交超过corePoolSize后，创建2个非核心线程执行，没有任务放进等待队列。

```
第一次提交：corePoolSize: 4     poolSize: 3     queueSize: 0
第二次提交：corePoolSize: 4     poolSize: 6     queueSize: 0
 8秒之后：corePoolSize: 4     poolSize: 4     queueSize: 0
```

keepAliveTime从5秒改成6秒，或者主线程等待时间8秒调到7秒，则非核心线程不回收。

```
第一次提交：corePoolSize: 4     poolSize: 3     queueSize: 0
第二次提交：corePoolSize: 4     poolSize: 6     queueSize: 0
 8秒之后：corePoolSize: 4     poolSize: 6     queueSize: 0
```

提交线程数（6）大于maxmumPoolSize（如改成5），则throw RejectedExecutionException。

```
第一次提交：corePoolSize: 4     poolSize: 3     queueSize: 0
Exception in thread "main" java.util.concurrent.RejectedExec
  at java.util.concurrent.ThreadPoolExecutor$AbortPolicy.rej
  at java.util.concurrent.ThreadPoolExecutor.reject(ThreadPo
  at java.util.concurrent.ThreadPoolExecutor.execute(ThreadP
  at pan.rufeng.demo.DemoThreadPoolExecuter.executeRunnable3
  at pan.rufeng.demo.DemoThreadPoolExecuter.main(DemoThreadP
```

- Test case 2: LinkedBlockingQueue without size limit, 6 tasks, corePoolSize 4, maxmumPoolSize 6, keepAliveTime 5秒

```
ThreadPoolExecutor tpe = new ThreadPoolExecutor(4, 6, 5, Tim
```

结果：第二次提交之后，2个任务放到队列等待。

```
第一次提交：corePoolSize: 4     poolSize: 3     queueSize: 0
第二次提交：corePoolSize: 4     poolSize: 4     queueSize: 2
 8秒之后：corePoolSize: 4     poolSize: 4     queueSize: 0
```

keepAliveTime改到8秒或者更大，对结果没影响，因为这里只用到核心线程。
maxmumPoolSize（如改成4）小于提交线程数（6），程序依然正常执行，结果同上。maxmumPoolSize限制失去意义（但不能小于corePoolSize）。

- Test case 3: ArrayBlockingQueue或LinkedBlockingQueue with size limit, 6 tasks, corePoolSize 4, maxmumPoolSize 6, keepAliveTime 5秒

```
ThreadPoolExecutor tpe = new ThreadPoolExecutor(4, 6, 5, Tim
ThreadPoolExecutor tpe = new ThreadPoolExecutor(4, 6, 5, Tim
```

结果：第二次提交超过corePoolSize后，创建1个非核心线程执行，1个任务放进等待队列。

```
第一次提交：corePoolSize: 4     poolSize: 3     queueSize: 0
第二次提交：corePoolSize: 4     poolSize: 5     queueSize: 1
 8秒之后：corePoolSize: 4     poolSize: 4     queueSize: 0
```

keepAliveTime从5秒改成6秒，或者主线程等待时间8秒调到7秒，则非核心线程不回收。

```
第一次提交：corePoolSize: 4     poolSize: 3     queueSize: 0
第二次提交：corePoolSize: 4     poolSize: 5     queueSize: 1
 8秒之后：corePoolSize: 4     poolSize: 5     queueSize: 0
```

可见，线程池的最大运行线程数为5，maxmumPoolSize设置需大于5，否则throw RejectedExecutionException。
将corePoolSize设为3，队列容量设为2，则可得如下结果。第二次提交任务时，创建1个非核心线程执行，剩下的2个任务进队列。

```
第一次提交：corePoolSize: 3     poolSize: 3     queueSize: 0
第二次提交：corePoolSize: 3     poolSize: 4     queueSize: 2
 8秒之后：corePoolSize: 3     poolSize: 3     queueSize: 0
```

具体验证例子也可参考：
https://blog.csdn.net/qq_25806863/article/details/71126867

Java Doc:
https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ThreadPoolExecutor.html

阅读导航：

[toc]

Build Custom JDK/JRE

```
#!/bin/sh
# Remove all not-needed files, see http://www.oracle.com/technet
# Documentation
rm -f jre/README.txt
rm -f jre/THIRDPARTYLICENSEREADME.txt
rm -f jre/THIRDPARTYLICENSEREADME-JAVAFX.txt
rm -f jre/Welcome.html
# optional files
rm -f jre/lib/ext/jfxrt.jar
rm -f jre/lib/ext/access-bridge.jar
rm -f jre/lib/ext/access-bridge-32.jar
rm -f jre/lib/ext/nashorn.jar
rm -f jre/bin/rmid.*
rm -f jre/bin/rmiregistry.*
rm -f jre/bin/tnameserv.*
rm -f jre/bin/keytool.*
rm -f jre/bin/kinit.*
rm -f jre/bin/klist.*
rm -f jre/bin/ktab.*
rm -f jre/bin/policytool.*
rm -f jre/bin/orbd.*
rm -f jre/bin/servertool.*
rm -f jre/bin/javaws.*
rm -f jre/lib/jfr.*
# can be deleted when private application runtime is used (like
# rm -f jre/bin/java.exe
# launch4j uses javaw.exe. So do NOT delete!
rm -f jre/bin/javacpl.exe
rm -f jre/bin/jabswitch.exe
rm -f jre/bin/java_crw_demo.dll
rm -f jre/bin/JavaAccessBridge-32.dll
rm -f jre/bin/JavaAccessBridge.dll
rm -f jre/bin/JAWTAccessBridge-32.dll
rm -f jre/bin/JAWTAccessBridge.dll
rm -f jre/bin/WindowsAccessBridge-32.dll
rm -f jre/bin/WindowsAccessBridge.dll
rm -f jre/bin/wsdetect.dll
rm -f jre/bin/deploy.dll
rm -f jre/bin/javacpl.cpl
rm -f jre/lib/deploy.jar
rm -f jre/lib/plugin.jar
rm -Rf /s /q jre/bin/dtplugin
rm -Rf /s /q jre/bin/plugin2
rm -Rf /s /q jre/lib/deploy
# JavaFX related
rm -f jre/lib/javafx.properties
rm -f jre/lib/jfxswt.jar
rm -f jre/bin/jfx*
```

```
rm -f jre/bin/decora_sse.dll
rm -f jre/bin/fxplugins.dll
rm -f jre/bin/glass.dll
rm -f jre/bin/glib-lite.dll
rm -f jre/bin/gstreamer-lite.dll
rm -f jre/bin/javafx_font.dll
```

Angular Helloworld

已经安装好Node.js， 通过以下命令可以新建一个angular项目。

```
npm i -g @angular/cli

ng new helloworld-ng

cd helloworld-ng
npm install

ng serve --open
```

https://angular.io/guide/quickstart

Angular Karma - Chrome have not captured in 60000 ms

给Google带偏了，各种改config各种试，花了大半天，给大家提供个思路。

这个错的root cause是Chrome启动不了。

为什么启动不了?
首先看下Chrome存不存在正不正常能不能启动！
Command line试一下。

我的情况是本机可以，运维机器不行，缺少相应权限，却又悲剧地默认运维机器的Chrome没问题！所以总以为问题出在其他地方！

开Debug可以看到Chrome启动的command line。首先检查下路径对不对。

```
20-Mar-2019 01:32:56  20 03 2019 01:32:56.381:INFO [karma]: Karm
20-Mar-2019 01:32:56  20 03 2019 01:32:56.381:INFO [launcher]: L
20-Mar-2019 01:32:56  20 03 2019 01:32:56.491:INFO [launcher]: S
20-Mar-2019 01:32:56  20 03 2019 01:32:56.492:DEBUG [temp-dir]:
20-Mar-2019 01:32:56  20 03 2019 01:32:56.493:DEBUG [launcher]:
20-Mar-2019 01:34:56  20 03 2019 01:34:56.511:WARN [launcher]: C
20-Mar-2019 01:34:58  20 03 2019 01:34:58.526:WARN [launcher]: C
20-Mar-2019 01:35:00  20 03 2019 01:35:00.542:WARN [launcher]: C
20-Mar-2019 01:35:00  20 03 2019 01:35:00.542:DEBUG [launcher]:

20-Mar-2019 01:35:00  20 03 2019 01:35:00.543:DEBUG [temp-dir]:
20-Mar-2019 01:35:00  20 03 2019 01:35:00.547:DEBUG [launcher]:
20-Mar-2019 01:35:00  20 03 2019 01:35:00.550:DEBUG [karma]: Run
20-Mar-2019 01:35:00  20 03 2019 01:35:00.551:DEBUG [launcher]:
20-Mar-2019 01:35:00  npm ERR! Test failed.  See above for more
```

Debug这样开。

karma.conf.js

```
logLevel: config.LOG_DEBUG,
```

可以这样加大启动时间：`captureTimeout:120000`（default 60000）。

另外，`browsers: ['Chrome']` 和 `browsers: ['ChromeHeadless']` 的区别是：ChromeHeadless是无弹窗模式。

自定义Browser，要注意自定义的名称是否对应（ChromeHeadless_test）。

```
browsers: ['ChromeHeadless_test'],
    customLaunchers: {
      ChromeHeadless_test: {
        base: 'ChromeHeadless',
        flags: [
          '--disable-extensions',
          '--no-sandbox',
          '--disable-web-security',
          '--no-proxy-server']
      }
    },
```

实际测试中剪掉一些feature会快两三秒，如非必要还是用默认就好，不喜弹窗可用 `browsers: ['ChromeHeadless']` 。

其他问题可以直接看launcher源代码，可能还快过Google，比较简单。

https://github.com/karma-runner/karma-chrome-launcher

Angular Tutorial里的测试坑

发现Angular Tutorial里根本没有提到Unit Test，其实我们每generate一个component，就会自动生成unit test spec，作为一个资深强迫症攻城狮，实在不能视而不见。

按照Angular Tutorial里的step，每增加一个功能，基本上会有一些test case会fail，虽然程序运行正常。主要是因为使用 `ng generate component` 时，会把新Module加入app.module.ts，但是却不会更新Unit Test。其他组件fail的原因也类似。

app-<component> is not a known element

`app-heros` 代码敲完后，页面显示正常，单元测试却报错如下。

```
Error: Template parse errors:
       'app-heros' is not a known element:
       1. If 'app-heros' is an Angular component, then verify t
       2. If 'app-heros' is a Web Component then add 'CUSTOM_EL
```

解决办法有两种。

直觉Mock才是好方法，不过有待证实。

因为后面会继续增加很多component，所以直接declare比较简单。

1. 导入HerosComponent

```
import { HerosComponent } from './heros/heros.component';

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent,
        HerosComponent
      ],
    }).compileComponents();
  }));
```

2. Mock HerosComponent

```
  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent,
        MockHerosComponent
      ],
    }).compileComponents();
  }));

@Component({
  selector: 'app-heros',
  template: ''
})
class MockHerosComponent{
}
```

其他Component

添加其他Component的时候也是一样，会遇到类似的错。

- app-heroes (HeroesComponent)
- app-messages (MessagesComponent)
- app-hero-detail (HeroDetailComponent)
- router-outlet (AppRoutingModule)
- app-dashboard (DashboardComponent)
- app-hero-search (HeroSearchComponent)

解决方法类似，如 `app-hero-detail` 报错，即需要declare `HeroDetailComponent` 。

```
Error: Template parse errors:
Can't bind to 'hero' since it isn't a known property of 'app-her
1. If 'app-hero-detail' is an Angular component and it has 'hero
2. If 'app-hero-detail' is a Web Component then add 'CUSTOM_ELEM
3. To allow any property add 'NO_ERRORS_SCHEMA' to the '@NgModul
</ul>
```

The module has not been imported

以下错误也一样，把相应Module放进declarations即可。

```
Failed: Component DashboardComponent is not part of any NgModule
```

router-outlet is not a known element

`router-outlet` 的情况有些特别，因为是RouterModule，所以AppRoutingModule需要放到imports下，而不是declarations里，与下面FormsModule一样。

```
Failed: Template parse errors:
'router-outlet' is not a known element:
1. If 'router-outlet' is an Angular component, then verify that
2. If 'router-outlet' is a Web Component then add 'CUSTOM_ELEMEN
  </nav>
  <!-- <app-heroes></app-heroes> -->
  [ERROR ->]<router-outlet></router-outlet>
  <app-messages></app-messages>
<!-- </div> -->
"): ng:///DynamicTestModule/AppComponent.html@9:2
```

但是AppRoutingModule是路由，import AppRoutingModule将导致子模块须引入更多其他不相关模块，所以不能作为解决方案。

还好Angular提供了RouterTestingModule。Import RouterTestingModule， 错误消失。

```
import { RouterTestingModule } from '../../../node_modules/@angu
  beforeEach(async(() => {
    TestBed.configureTestingModule({
      imports: [RouterTestingModule, HttpClientTestingModule],
      declarations: [HeroSearchComponent]
    })
      .compileComponents();
  }));
```

Can't bind to 'routerLink'

`routerLink` 类似于 `router-outlet` 。

```
Failed: Template parse errors:
Can't bind to 'routerLink' since it isn't a known property of 'a
<ul class="heroes">
  <li *ngFor="let hero of heroes">
    <a [ERROR ->]routerLink="/detail/">
      <span class="badge"></span>
    </a>"): ng:///DynamicTestModule/HeroesComponent.html@13:7
```

Solution: Imports RouterTestingModule.

```
import { RouterTestingModule } from '@angular/router/testing';

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      imports: [RouterTestingModule.withRoutes([{ path: 'heroes'
        HttpClientTestingModule ],
      declarations: [ HeroesComponent ]
    })
    .compileComponents();
  }));
```

No base href set

这个错会在刚开始用AppRoutingModule时遇到。因为我们很自然会想到要在test case里import AppRoutingModule。

```
Error: No base href set. Please provide a value for the APP_BASE
```

TestBed.configureTestingModule加以下configure可以解决。

```
providers: [{provide: APP_BASE_HREF, useValue: '/' }]
```

但更简单的做法是把测试里的AppRoutingModule替换为RouterTestingModule。

```
beforeEach(async(() => {
  TestBed.configureTestingModule({
    declarations: [
      AppComponent,
      HeroesComponent,
      MessagesComponent,
      HeroDetailComponent,
      DashboardComponent,
      HeroSearchComponent
    ],
    imports: [
      FormsModule,
      // AppRoutingModule
      RouterTestingModule
    ]
    // providers: [{provide: APP_BASE_HREF, useValue: '/' }]
  }).compileComponents();
}));
```

Can't bind to 'ngModel'

页面使用到 `ngModel` 时，控件绑定需要import FormsModule。这个教程里也有提到，但是只是针对app.module.ts，其实Test Case里也要。

```
Failed: Template parse errors:
Can't bind to 'ngModel' since it isn't a known property of 'inpu
  <div>
    <label>name:
      <input [ERROR ->][(ngModel)]="hero.name" placeholder="name
    </label>
  </div>
"): ng:///DynamicTestModule/HeroDetailComponent.html@6:13
```

Solution: Imports FormsModule.

```
  beforeEach(async(() => {
    TestBed.configureTestingModule({
      imports: [ FormsModule,
        RouterTestingModule.withRoutes([{ path: 'detail/:id', co
        HttpClientTestingModule ],
      declarations: [ HeroDetailComponent ]
    })
    .compileComponents();
  }));
```

Error: StaticInjectorError

这个错出现在引入 `angular-in-memory-web-api` 模拟server response的时候。HeroService需要注入HttpClient。

```
Error: StaticInjectorError(DynamicTestModule)[HttpClient]:
  StaticInjectorError(Platform: core)[HttpClient]:
    NullInjectorError: No provider for HttpClient!
```

Solution: Imports HttpClientTestingModule.

```
import { HttpClientTestingModule } from '@angular/common/http/te

imports: [ FormsModule,
        RouterTestingModule.withRoutes([{ path: 'heroes', compon
        HttpClientTestingModule ],
```

示例代码

app.component.spec.ts

```
import { TestBed, async } from '@angular/core/testing';
import { AppComponent } from './app.component';
import { HeroesComponent } from './heroes/heroes.component';
import { MessagesComponent } from './messages/messages.component
// import { AppRoutingModule } from './app-routing.module';

import { HeroDetailComponent } from './hero-detail/hero-detail.c

import { FormsModule } from '@angular/forms';
// import {APP_BASE_HREF} from '@angular/common';
import { DashboardComponent } from './dashboard/dashboard.compon
import { HeroSearchComponent } from './hero-search/hero-search.c
import { RouterTestingModule } from '../../node_modules/@angular

describe('AppComponent', () => {
  beforeEach(async(() => {
    TestBed.configureTestingModule({
      declarations: [
        AppComponent,
        HeroesComponent,
        MessagesComponent,
        HeroDetailComponent,
        DashboardComponent,
        HeroSearchComponent
      ],
      imports: [
        FormsModule,
        // AppRoutingModule
        RouterTestingModule
      ]
      // providers: [{provide: APP_BASE_HREF, useValue: '/' }]
    }).compileComponents();
  }));

  it('should create the app', () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app).toBeTruthy();
  });

  it(`should have as title 'Tour of Heroes'`, () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.debugElement.componentInstance;
    expect(app.title).toEqual('Tour of Heroes');
  });

  it('should render title in a h1 tag', () => {
    const fixture = TestBed.createComponent(AppComponent);
```

```
    fixture.detectChanges();
    const compiled = fixture.debugElement.nativeElement;
    expect(compiled.querySelector('h1').textContent).toContain('
  });
});

// @Component({
//   selector: 'app-heros',
//   template: ''
// })
// class MockHeroesComponent{
// }
```

dashboard.component.spec.ts

```
import { async, ComponentFixture, TestBed } from '@angular/core/

import { DashboardComponent } from './dashboard.component';

import { RouterTestingModule } from '@angular/router/testing';
import { HttpClientTestingModule } from '@angular/common/http/te
import { HeroSearchComponent } from '../hero-search/hero-search.

describe('DashboardComponent', () => {
  let component: DashboardComponent;
  let fixture: ComponentFixture<DashboardComponent>;

  beforeEach(async(() => {
    TestBed.configureTestingModule({
      imports: [RouterTestingModule.withRoutes([{ path: 'dashboa
        HttpClientTestingModule],
      declarations: [DashboardComponent, HeroSearchComponent]
    })
      .compileComponents();
  }));

  beforeEach(() => {
    fixture = TestBed.createComponent(DashboardComponent);
    component = fixture.componentInstance;
    fixture.detectChanges();
  });

  it('should create', () => {
    expect(component).toBeTruthy();
  });
});
```

Reference

https://angular.io/tutorial
https://stackblitz.com/angular/xedbpmlyaag?file=src%2Fapp%2Fapp.module.ts
https://angular.io/api/router/testing/RouterTestingModule

Angular只跑（或只不跑）某一个Test Case

只跑(Only Run)

Add f before `describe` or `it`, i.e.

```
fdescribe
fit
```

只不跑(Exclude)

Add x before `describe` or `it`, i.e.

```
xdescribe
xit
```

换一台电脑打开GitBook项目居然运行不了， `gitbook serve` 和 `gitbook build` 都报错。

Error: ENOENT: no such file or directory, stat 'C:***demo_book_book\gitbook\gitbook-plugin-fontsettings\fontsettings.js'

原来是一个Bug（Vesion：3.2.3）。

https://github.com/GitbookIO/gitbook/issues/1309

解决办法如下。

用户目录下找到以下文件。
\\.gitbook\versions\3.2.3\lib\output\website\copyPluginAssets.js

Replace all
`confirm: true`
with
`confirm: false`

JavaScript if(变量) 到底什么时候成立

JavaScript实在是神通广大，虽然写了很多年，一些简单的代码有时写着写着还是忍不住会怀疑。

比如if(variable) 这样的语句到底什么时候成立，具体是怎么判断的？

今天忍不住再来测试一下。

结论

返回false的变量值：

- false
- 0
- 空字符串
- undefined
- null
- NaN

返回true的变量值：

- true
- 1
- 非空字符串
- 对象
- 数组

非空字符串包括空格（`' '`），`'false'`，`'0'`。

空对象和空数组都返回true。

测试代码

```
let count = 0;
let func = (tt) => {
    count++;
    if (tt) {
        console.log(count + '. ' + tt + ' is true');
    } else {
        console.log(count + '. ' + tt + ' is false');
    }
};

let tt;
func(tt);
tt = 1;
func(tt);
tt = 0;
func(tt);
tt = true;
func(tt);
tt = false;
func(tt);
tt = 'true';
func(tt);
tt = 'false';
func(tt);
tt = '0';
func(tt);
tt = '';
func(tt);
tt = ' ';
func(tt);
tt = null;
func(tt);
tt = NaN;
func(tt);
tt = {};
func(tt);
tt = [];
func(tt);
```

输出结果

```
1. undefined is false
2. 1 is true
3. 0 is false
4. true is true
5. false is false
6. true is true
7. false is true
8.  is false
9.   is true
10. null is false
11. NaN is false
12. [object Object] is true
13.  is true
```

其他逻辑运算

From: w3cshool

NOT

与逻辑 OR 和逻辑 AND 运算符不同的是，逻辑 NOT 运算符返回的一定是 Boolean 值。

逻辑 NOT 运算符的行为如下：

- 如果运算数是对象，返回 false
- 如果运算数是数字 0，返回 true
- 如果运算数是 0 以外的任何数字，返回 false
- 如果运算数是 null，返回 true
- 如果运算数是 NaN，返回 true
- 如果运算数是 undefined，发生错误

AND

逻辑 AND 运算的运算数可以是任何类型的，不止是 Boolean 值。

如果某个运算数不是原始的 Boolean 型值，逻辑 AND 运算并不一定返回 Boolean 值：

- 如果一个运算数是对象，另一个是 Boolean 值，返回该对象。
- 如果两个运算数都是对象，返回第二个对象。
- 如果某个运算数是 null，返回 null。
- 如果某个运算数是 NaN，返回 NaN。
- 如果某个运算数是 undefined，发生错误。

OR

与逻辑 AND 运算符相似，如果某个运算数不是 Boolean 值，逻辑 OR 运算并不一定返回 Boolean 值：

- 如果一个运算数是对象，并且该对象左边的运算数值均为 false，则返回该对象。
- 如果两个运算数都是对象，返回第一个对象。
- 如果最后一个运算数是 null，并且其他运算数值均为 false，则返回 null。
- 如果最后一个运算数是 NaN，并且其他运算数值均为 false，则返回 NaN。
- 如果某个运算数是 undefined，发生错误。

JavaScript设计模式 - Strategy

专职写了几个月，虽然开始有些凌乱，现在基本上也认同：JavaScript是一门优雅的语言。

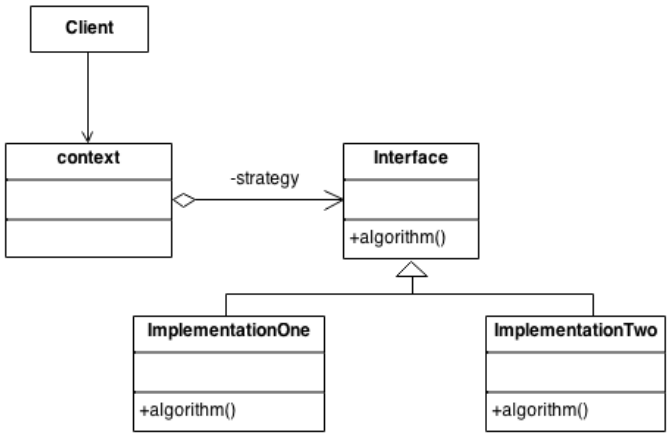
从设计模式的角度，应该能更好地理解JavaScript的不同，改善既有代码。

那就先从比较常用的Strategy模式开始。

Strategy Pattern

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from the clients that use it.

(From https://sourcemaking.com/design_patterns/strategy)



JavaScript Example

- 定义一组Strategy, 确保可互相替换
- 通过替换Strategy改变程序的行为

```
const HelloWork = function (travelStrategy) {
    this.travelStrategy = travelStrategy;

    this.setTravelStrategy = function (travelStrategy) {
        this.travelStrategy = travelStrategy;
    };
    this.travel = function () {
        let strategy = this.travelStrategy();
        console.log(`${strategy}上班`);
    };
};

const walking = () => {
    return '我穷我走路';
};
const bike = () => {
    return '我环保我开新能源机动车（小电瓶）';
};
const subway = () => {
    return '我土豪我搭几个亿的地铁';
};

const hw = new HelloWork(walking);
hw.travel();

hw.setTravelStrategy(bike);
hw.travel();

hw.setTravelStrategy(subway);
hw.travel();
```

Output:

```
我穷我走路上班
我环保我开新能源机动车（小电瓶）上班
我土豪我搭几个亿的地铁上班
```

JavaScript进阶 - Promise

Promise，中文即承诺，Promise承诺的是，未来将会按照指定的顺序执行已定义的异步操作。

Promise实质上是一个代表异步操作最终完成或失败的对象。它主要通过维护程序的内部状态来实现对流程的控制。

Promise的三种状态

Promise只能处于以下三种状态：

- fulfilled
- rejected
- pending

当操作执行成功时，调用resolve函数将promise状态改为fulfilled，失败时，调用reject函数将promise状态改为rejected。
程序正在运行时，状态为pending，当抛出异常，同样返回rejected。

Promise定义语法如下，支持Promise的方法必须return一个Promise对象。

```
new Promise( function(resolve, reject) {...} /* executor */  );
```

Promise的三个方法

- then
- catch
- finally

以上三个方法都会返回一个新的Promise，所以Promise支持链式调用。

多次调用then，Promise会确保前面的异步操作完成之后再执行后面的操作，按插入顺序独立运行。

Promise链式调用里，一个catch即可捕获Promise所有代码块里的异常。

Promise运行完之后要调用的代码可以放在finally里，无论最终状态是成功或失败，它都会被调用到。

Promise示例代码

```
const myPromise1 = function (param) {
    return new Promise((resolve, reject) => {
        console.log('myPromise1 is running');
        setTimeout(() => {
            if (param) {
                resolve('promise1');
            } else {
                reject(new Error('Oops!'));
            }
        }, 0);
    });
};

const myPromise2 = function (param) {
    return new Promise((resolve, reject) => {
        console.log(`myPromise2 Param is ${param}`);
        // throw new Error('Oops!'); //5
        // reject(new Error('Oops!'));  //6
        // dummyFunc(); //7 not defined
        setTimeout(() => resolve('promise2'), 50);
    }).catch((err) => {
        console.log('catch Exception in myPromise2');
        console.log(err);
        throw new Error('Oh No!'); //8
    });
};

const myPromise3 = function (param) {
    setTimeout(() => {
        console.log(`myPromise3 Param is ${param}`);
        throw new Error('Oops!'); //1
    }, 450); //2
};

const myPromise4 = function (param) {
    return new Promise((resolve, reject) => {
        console.log(`myPromise4 Param is ${param}`);
        setTimeout(() => resolve('promise4'), 50);
    });
};

myPromise1(1) //4
    .then(myPromise2)
    .then(myPromise3)
    .then(myPromise4)
    .then(ret => {
        console.log(`final block return ${ret}`);
    }).catch(err => {//3
```

```
            console.log('catch Exception in the end');
            console.log(err);
        }).then(() => {
            console.log('then after catch');
        }).finally(() => myPromise4('finally')).then(() => {
            console.log('then after finally');
            done();
        });
```

Promise嵌套

Promise嵌套的必须是正确定义的Promise方法（return Promise对象），否则异步操作的结果（次序）无法保证。

上述示例代码的运行结果如下：

```
myPromise1 is running
myPromise2 Param is promise1
myPromise4 Param is undefined
final block return promise4
then after catch
myPromise4 Param is finally
then after finally

myPromise3 Param is promise2
```

这个结果存在以下问题：

1. myPromise4并没有拿到myPromise3作为参数（undefined）
2. myPromise3并没有出现在期望的位置（而是最后）
3. myPromise3没有throw Error

因为myPromise3并不是Promise方法，而只是一个普通的异步方法。当myPromise2执行完之后，myPromise3开始执行，同时myPromise4也开始执行，而不是等待myPromise3完成之后。 myPromise4之后的操作都按顺序执行，myPromise3则同时独自完成自身逻辑。

myPromise3没有throw Error（//1），是由于主程序在此之前就已经运行完毕退出了。

将myPromise3的timeout时间从450改为50（//2），则可成功抛出异常。

```
myPromise1 is running
myPromise2 Param is promise1
myPromise4 Param is undefined
myPromise3 Param is promise2
final block return promise4
then after catch
myPromise4 Param is finally

     Uncaught Error: Oops!
      at Timeout.setTimeout [as _onTimeout] (test\demo\promise.s

then after finally
(node:25368) UnhandledPromiseRejectionWarning: TypeError: Cannot
    at C:\swdtools\node-v10.8.0-win-x64\node_modules\mocha\lib\r
    at done (C:\swdtools\node-v10.8.0-win-x64\node_modules\mocha
    at C:\swdtools\node-v10.8.0-win-x64\node_modules\mocha\lib\r
    at myPromise1.then.then.then.then.catch.then.finally.then (C
(node:25368) UnhandledPromiseRejectionWarning: Unhandled promise
(node:25368) [DEP0018] DeprecationWarning: Unhandled promise rej
```

Promise异常处理

UnhandledPromiseRejection

假如Promise没有添加任何catch代码块，程序出错（rejected）时将会报UnhandledPromiseRejection。

开发过程中遇到此错误说明Promise相关代码可能还存在问题。

在示例中删除catch代码块（//3），将myPromise1参数改为0（//4）则重现此错误。

```
myPromise1 is running
myPromise4 Param is finally
(node:26412) UnhandledPromiseRejectionWarning: Error: Oops!
    at Timeout.setTimeout [as _onTimeout] (C:\Users\pan\workspac
    at ontimeout (timers.js:424:11)
    at tryOnTimeout (timers.js:288:5)
    at listOnTimeout (timers.js:251:5)
    at Timer.processTimers (timers.js:211:10)
(node:26412) UnhandledPromiseRejectionWarning: Unhandled promise
(node:26412) [DEP0018] DeprecationWarning: Unhandled promise rej
```

隐含try...catch

Promise定义（executor）里的代码无需显式添加try...catch代码块，错误会被自动捕获成一个rejection。

所以Promise里以下代码的效果是一样的。

```
// throw new Error('Oops!'); //5
// reject(new Error('Oops!'));  //6
```

Rethrow

可以在链式调用的任何一个Promise里catch异常，进行必要的处理后重新throw出来。

启用myPromise2里的dummyFunc()（//7），运行结果如下。
可以看到Error两次被catch，而且不再运行后面then里面的myPromise3和myPromise4（catch之后没影响）。

```
myPromise1 is running
myPromise2 Param is promise1
catch Exception in myPromise2
ReferenceError: dummyFunc is not defined
    at Promise (C:\Users\pan\workspace\nodejs\autotest-node\test
    at new Promise (<anonymous>)
    at myPromise2 (C:\Users\pan\workspace\nodejs\autotest-node\t

    at myPromise1.then.ret (C:\Users\pan\workspace\nodejs\autote
catch Exception in the end
Error: Oh No!
    at Promise.catch (C:\Users\pan\workspace\nodejs\autotest-nod
then after catch
myPromise4 Param is finally
then after finally
```

Comment掉 `throw new Error('Oh No!');` （//8），则结果如下。
因为Error已经在myPromise2里catch和处理，所以后面的代码都正常运行。

```
myPromise1 is running
myPromise2 Param is promise1
catch Exception in myPromise2
ReferenceError: dummyFunc is not defined
    at Promise (C:\Users\pan\workspace\nodejs\autotest-node\test
    at new Promise (<anonymous>)
    at myPromise2 (C:\Users\pan\workspace\nodejs\autotest-node\t
    at myPromise1.then.ret (C:\Users\pan\workspace\nodejs\autote
myPromise4 Param is undefined
final block return promise4
then after catch
myPromise4 Param is finally
then after finally

myPromise3 Param is undefined
```

怎么测试Promise

Promise方法的测试和普通方法一样，只是返回对象比较特殊。

测试的难点在于，在调用程序里，如何stub Promise方法。

sinon为Promise提供了如下支持。

测试正常执行（resolve）条件下的代码块：

```
sinon.stub(s, 'sendMessage').resolves('data');
```

测试catch里（reject）的代码块：

```
sinon.stub(s, 'sendMessage').returns(Promise.reject(new Error('O
```

Node.js Bunyan输出JSON日志

日志应该是结构化的，处理日志的主要应该是机器而不是人，这就是日志输出为JSON的好处，而Bunyan即是一个简单快速的JSON日志库。

正式Demo可参考Github：
https://github.com/prufeng/autotest-node/tree/master/demo

Install

```
npm install bunyan
```

Helloworld

测试代码

```
//app.js
var bunyan = require('bunyan');
var log = bunyan.createLogger({name:'app'});
log.info('hello world');
log.warn({key: 'test'}, 'hello world');
log.error('hello world');
```

输出JSON

```
{"name":"app","hostname":"PC70358896","pid":12248,"level":30,"ms
{"name":"app","hostname":"PC70358896","pid":12248,"level":40,"ke
{"name":"app","hostname":"PC70358896","pid":12248,"level":50,"ms
```

输出到文件

```
var log2file = bunyan.createLogger({
  name: 'log2file',
  streams:[{
    // type: 'file', //default type
    path: './logs/bunyan.log',
  }]
});
log2file.info('hello world');
log2file.warn({key: 'test'}, 'hello world');
log2file.error('hello world');
```

输出到轮转文件

可定义多个Stream，rotating-file昨天的log自动加后缀.0，前天加后缀.1，以此类推。

另外，测试过程中发现要事先建立logs文件夹，否则会fail。

```
var log2file = bunyan.createLogger({
  name: 'log2file',
  streams:[{
    // type: 'file',  //default type
    path: './logs/bunyan.log',
  },
  {
    type: 'rotating-file',
    path: './logs/bunyan-rf.log',
    period: '1d', //daily rotation
    count: 3  //keep 3 backup
  },
  {
    type: 'rotating-file',
    level: 'error', //error log, default info
    path: './logs/bunyan-rf-error.log',
    period: '1d',
    count: 3
  }]
});
log2file.info('hello world');
log2file.warn({key: 'test'}, 'hello world');
log2file.error('hello world');
```

express-bunyan-logger

Install

```
npm i express-bunyan-logger
```

测试代码

```
//app.js
app.use(require('express-bunyan-logger')(express));

var express = {
    name: 'bunyan-express',
    // format: ":remote-address - :user-agent[major] custom logg
    streams: [
        {
            type: 'rotating-file',
            path: './logs/bunyan-rf-express.log',
            period: '1d',
            count: 3
        }
    ]
};
```

输出

```
{"name":"bunyan-express","hostname":"PC70358896","pid":16780,"re
```

参考

https://github.com/trentm/node-bunyan
https://github.com/villadora/express-bunyan-logger
https://medium.com/@bansalnagesh/better-logging-with-bunyan-eff940d29956

log4js-node是log4js的Node.js版本。

log4js-node的使用比较简单，以下内容主要部分来自官方文档。

https://github.com/log4js-node/log4js-node

Installation

```
npm install log4js
```

Usage

注意：log4js默认category的level是OFF，所以以下程序不会有输出。

```
var log4js = require('log4js');
var logger = log4js.getLogger();
logger.level = 'debug';
logger.debug("Some debug messages");
```

以下程序将default category的level改为error。

```
const log4js = require('log4js');
log4js.configure({
  appenders: { cheese: { type: 'file', filename: 'cheese.log' }
  categories: { default: { appenders: ['cheese'], level: 'error'
});

const logger = log4js.getLogger('cheese');
logger.trace('Entering cheese testing');
logger.debug('Got cheese.');
logger.info('Cheese is Comté.');
logger.warn('Cheese is quite smelly.');
logger.error('Cheese is too ripe!');
logger.fatal('Cheese was breeding ground for listeria.');
```

成功输出error以上级别的log。

```
[2010-01-17 11:43:37.987] [ERROR] cheese - Cheese is too ripe!
[2010-01-17 11:43:37.990] [FATAL] cheese - Cheese was breeding g
```

Example

https://github.com/log4js-node/log4js-example

1. log4js.configure()

configure可以接收一个JSON文件（log4js.json）作为参数。

2. log4js.json

定义了3种category、4种appender。其中errors是logLevelFilter，输出到errorFile，就是所有error信息会单独输出到errors.log文件中。

```
{
    "appenders": {
      "access": {
        "type": "dateFile",
        "filename": "log/access.log",
        "pattern": "-yyyy-MM-dd",
        "category": "http"
      },
      "app": {
        "type": "file",
        "filename": "log/app.log",
        "maxLogSize": 10485760,
        "numBackups": 3
      },
      "errorFile": {
        "type": "file",
        "filename": "log/errors.log"
      },
      "errors": {
        "type": "logLevelFilter",

        "level": "ERROR",
        "appender": "errorFile"
      }
    },
    "categories": {
      "default": { "appenders": [ "app", "errors" ], "level": "D
      "http": { "appenders": [ "access"], "level": "DEBUG" }
    }
  }
```

3. log4js调用

根据category决定输出文件，getLogger('http')输出到access.log，其他情况下输出到app.log（因为category非http即default），error自动输出到errors.log文件。

```
var log4js = require('log4js');
var log = log4js.getLogger("app");

app.use(log4js.connectLogger(log4js.getLogger("http"), { level:

log.error("Something went wrong:", err);
```

动态配置

1. 不同的log4js.json

因为log4js.configure支持json文件作为参数，可以为Development和Production定义不同的配置文件。
如配置文件放在config目录下，区分dev和prod。

config/dev/log4js.json
config/prod/log4js.json

2. 调用方法

根据ENV值选择调用的配置文件，即运行前set ENV=prod，default是dev。

```
let env = process.env.ENV || 'dev';
let logConfigPath = './config/' + env + '/log4js.json';
let logConfig = require(logConfigPath);
log4js.configure(logConfig);
```

相关文章

- Node.js Log4js输出JSON日志
- Node.js Log4js输出自定义格式和变量
- Node.js Log4js写Log的同时想做点什么，可以自己写Appender

Node.js Log4js写Log的同时想做点什么，可以自己写Appender

log4js-node集成appender用的是Listener模式， 所以可以方便地添加自定义的appender，这样就能在写log的同时做点额外的事情，比如把log发送到自己设计好的接口。

Optional Appenders

以下是log4js-node当前提供的可选appender。

- gelf
- hipchat
- logFaces-HTTP
- logFaces-UDP
- loggly
- logstashHTTP
- logstashUDP
- mailgun
- rabbitmq
- redis
- slack
- smtp

Writing Appender

My Appender

可以直接copy一个现成的appender来改，这里是copy dateFile，实现部分只是加 `Process by myAppender -` 前缀输出，而且只处理ERROR以上的log。

log4js-myAppender.js

```
'use strict';

const os = require('os');

const eol = os.EOL || '\n';

/**
 * TODO: incomplete example.
 * @filename base filename.
 * @pattern the format that will be added to the end of filename
 *          also used to check when to roll files - defaults to
 * @layout layout function for log messages - defaults to basicL
 * @timezoneOffset optional timezone offset in minutes - default
 */
function appender(
  filename,
  pattern,
  layout,
  options,
  timezoneOffset
) {
  const app = function (logEvent) {
    if (logEvent.level.isGreaterThanOrEqualTo('ERROR')){
      console.log('Process by myAppender - ' + layout(logEvent,
    }

    // console.log(config.level);
  };

  app.shutdown = function (complete) {
    // complete();
  };

  return app;
}

function configure(config, layouts) {
  let layout = layouts.basicLayout;

  if (config.layout) {
    layout = layouts.layout(config.layout.type, config.layout);
  }

  if (!config.alwaysIncludePattern) {
    config.alwaysIncludePattern = false;
  }

  return appender(
```

```
    config.filename,
    config.pattern,
    layout,
    config,
    config.timezoneOffset
  );
}

module.exports.configure = configure;
```

log4js.configure

`type` 填写对应appender相对于项目根目录的路径，其他参数此例只用到 `level` 。

```
        "errors": {
            "type": './demo/log4js-myAppender',
            'level': 'ERROR'
        }
```

Appender Type

需要小心的是插件的加载机制。
Log4js configure里appender type需要填写相对于project root的路径。

已有的可选插件路径不需要这样写是因为作为独立module发布了。

具体加载顺序如下：

1. The core appenders: `require('./appenders/' + type)`
2. node_modules: `require(type)`
3. relative to the main file of your application: `require(path.dirname(require.main.filename) + '/' + type)`
4. relative to the process' current working directory: `require(process.cwd() + '/' + type)`

参考

https://github.com/log4js-node/log4js-node/blob/master/docs/writing-appenders.md

Node.js Log4js输出JSON日志

Log4js Json Layout

代码：

```
const log4js = require('log4js');

log4js.addLayout('json', function(config) {
  return function(logEvent) { return JSON.stringify(logEvent) +
});

log4js.configure({
  appenders: {
    out: { type: 'stdout', layout: { type: 'json', separator: ',
  },
  categories: {
    default: { appenders: ['out'], level: 'info' }
  }
});

const logger = log4js.getLogger('json-test');
logger.info('this is just a test');
logger.error('of a custom appender');
logger.warn('that outputs json');
log4js.shutdown(() => {});
```

输出：

```
{"startTime":"2017-06-05T22:23:08.479Z","categoryName":"json-tes
{"startTime":"2017-06-05T22:23:08.483Z","categoryName":"json-tes
{"startTime":"2017-06-05T22:23:08.483Z","categoryName":"json-tes
```

Issue

发现 `startTime` 的时间格式与普通layout不一致，输出的并不是local时间。GitHub上给他们提了一个Issue。

https://github.com/log4js-node/log4js-node/issues/814

参考

https://github.com/log4js-node/log4js-node/blob/master/docs/layouts.md

Node.js Log4js输出自定义格式和变量

Log4js输出自定义格式，或者想在log里添加自己定义的变量，如显示当前用户名，可以通过pattern layout来实现。

Log4js Pattern Layout

- 设置占位符如 `%X{user}`
- 在代码里调用 `addContext('user','userName')` 更新变量

```
log4js.configure({
  appenders: {
    out: { type: 'stdout', layout: {
      type: 'pattern',
      pattern: '%d %p %c %X{user} %m%n'
    }}
  },
  categories: { default: { appenders: ['out'], level: 'info' } }
});
const logger = log4js.getLogger();
logger.addContext('user', 'charlie');
logger.info('doing something.');
```

This would output:

```
2017-06-01 08:32:56.283 INFO default charlie doing something.
```

如何优雅地addContext

思路是利用express middleware，截取request获得参数信息，调用addContext()设置到log4js后，再转发到router的其他请求。这样，在该router里的所有logger都可以格式化地print出我们定义的参数。

这里有个简单Demo，其他AOP方法尚待研究。
https://github.com/prufeng/autotest-node

参考

https://github.com/log4js-node/log4js-node/blob/master/docs/layouts.md

Node.js RESTful API和Express单元测试

本文主要研究Node.js RESTful API和Express框架的单元测试。

关于Mocha和Istanbul的详细使用，请参考：使用Mocha和Istanbul实现Node.js单元测试和覆盖率

Github: https://github.com/prufeng/autotest-node

Install

```
npm i -D nyc
npm i -D mocha
npm i -D chai
npm i -D chai-http
```

Unit Test Spec

创建单元测试文件app.spec.js。
chai主要用来测试HTTP request，done用来保证异步测试完成之后mocha才返回结果。

```
let chai = require('chai');
let chaiHttp = require('chai-http');
let app = require('../app');

let should = chai.should();
chai.use(chaiHttp);

describe('App', () => {
    it('should respond status 200', (done) => {
      chai.request(app)
          .get('/')
          .end((err, res) => {
                res.should.have.status(200);
            done();
          });
    });
    it('should GET the users response', (done) => {
        chai.request(app)
            .get('/users')
            .end((err, res) => {
                  res.should.have.status(200);
                  res.text.should.equal('respond with a resource
              done();
            });
      });

      it('should respond status 404', (done) => {
        chai.request(app)
            .get('/wrongUrl')
            .end((err, res) => {
                  res.should.have.status(404);
              done();
            });
      });
});
```

Run Test

package.json

`-a` 用来覆盖所有代码文件， `--recursive` 表示包含子目录。

```
  "scripts": {
    "start": "node ./bin/www",
    "test": "nyc -a mocha --recursive"
  }
```

npm test

```
npm test
```

```
  App
GET / 200 527.550 ms - 170
    v should respond status 200 (550ms)
GET /users 200 1.022 ms - 23
    v should GET the users response
GET /wrongUrl 404 11.877 ms - 2583
    v should respond status 404

  Calculator
    add
      v add(1,2) should return 3
    minus
      v minus(1,2) should return -1

  Array
    #indexOf()
      v should return -1 when the value is not present


  6 passing (588ms)

-----------------|----------|----------|----------|----------|--
File             |  % Stmts | % Branch |  % Funcs |  % Lines | U
-----------------|----------|----------|----------|----------|--
All files        |      100 |       50 |      100 |      100 |
 autotest        |      100 |       50 |      100 |      100 |
  app.js         |      100 |       50 |      100 |      100 |
  calc.js        |      100 |      100 |      100 |      100 |
 autotest/routes |      100 |      100 |      100 |      100 |
  index.js       |      100 |      100 |      100 |      100 |
  users.js       |      100 |      100 |      100 |      100 |
-----------------|----------|----------|----------|----------|--
```

Reference

https://scotch.io/tutorials/test-a-node-restful-api-with-mocha-and-chai
https://github.com/samuxyz/bookstore
https://github.com/chaijs/chai

Node.js Sinon测试替身

测试替身

测试替身(Test Double)，顾名思义，即测试时用来代替系统中某一部分的模拟技术的统称。

测试替身的作用

- 隔离被测代码
- 加速执行测试
- 使执行变得确定
- 模拟特殊情况
- 访问隐藏信息

上述列表copy from：https://yq.aliyun.com/articles/118887

测试替身的类型

- Dummy objects are passed around but never actually used. Usually they are just used to fill parameter lists.
- Fake objects actually have working implementations, but usually take some shortcut which makes them not suitable for production (an in memory database is a good example).
- Stubs provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test.
- Spies are stubs that also record some information based on how they were called. One form of this might be an email service that records how many messages it was sent.
- Mocks are what we are talking about here: objects pre-programmed with expectations which form a specification of the calls they are expected to receive.

上述分类copy from大神blog：
https://martinfowler.com/articles/mocksArentStubs.html

Sinon单元测试替身

Node.js的单元测试替身技术，Sinon应该是主流，下面结合实例分析理解下。

例子主要来源于《JavaScript测试驱动开发》，不过书里的Sinon版本已经过时了，无法运行，所以这里是改进版。也用到Mocha和Chai。

https://github.com/prufeng/autotest-node/blob/master/test/pam/stockfetch.spec.js

```
it('read should invoke error handler for invalid file', func
    var onError = function (err) {
        expect(err).to.be.eql('Error reading file: dummyInva
        done();
    };
    var stub = sinon.stub(fs, 'readFile');
    stub.yields(new Error('Failed to read file'));

    stockFetch.readTickerFile('dummyInvalidFile', onError);
    stub.restore();
});

it('read should invoke processTickers for valid file', funct
    var rawData = '601169\n002146\n601009'
    var parsedData = ['601169', '002146', '601009'];

    sinon.stub(fs, 'readFile').callsFake(function(fileName,
        callback(null, rawData);
    });

    sinon.stub(stockFetch, 'parseTickers').withArgs(rawData)

    sinon.stub(stockFetch, 'processTickers').callsFake(funct
        expect(data).to.be.eql(parsedData);
    });

    stockFetch.readTickerFile('dummyValidFile', function(err
});


it('read should return error if given file is empty', functi
    var onError = function(err){
        expect(err).to.be.eql('File dummyEmptyFile has inval
        done();
    };

    sinon.stub(stockFetch, 'parseTickers').withArgs('').retu
    sinon.stub(fs, 'readFile').callsFake(function(fileName,
        callback(null, '');
    });

    stockFetch.readTickerFile('dummyEmptyFile', onError);
});


it('processTickers should call getPrice for each ticker symb
    const stockFetchMock = sinon.mock(stockFetch);
    stockFetchMock.expects('getPrice').withArgs('601169');
```

```
        stockFetchMock.expects('getPrice').withArgs('002146');
        stockFetchMock.expects('getPrice').withArgs('601009');

        stockFetch.processTickers(['601169', '002146', '601009']
        stockFetchMock.verify();
    });

    it('processTickers should call getPrice for each ticker symb
        const stockFetchSpy = sinon.spy(stockFetch, 'getPrice');

        stockFetch.processTickers(['601169', '002146', '601009']

        expect(stockFetchSpy.firstCall.calledWithExactly('601169
        expect(stockFetchSpy.secondCall.calledWithExactly('00214
        expect(stockFetchSpy.thirdCall.calledWithExactly('601009
    });
```

Dummy

如 `dummyInvalidFile` ， `dummyValidFile` 和 `dummyEmptyFile` ，该参数在程序里实际上并没有参与执行，因为 `fs.readFile` 方法已经被Stub了。

此即没有真正用到但又需要填充的情形。建议起个好点的名字，最好一目了然。

Stub

Stub并不是真正的实现，主要用来在调用时快速返回预设数据。

Stub相当于把目标替换了，原本的方法不会再执行。

通常使用Stub来预设依赖数据，控制程序的行为，使得我们可以一次只测试一个我们关心的特定路径，实现代码隔离。

比如 `fs.readFile` 的Stub，使得程序无需真正地进行文件操作，同时可以赋予不同的返回数据，分别测试（定义）了目标方法在读文件出错，读正常文件和读空文件情形下的行为。

Stub `parseTickers` 和 `processTickers` ，则使得该Test Case可以专注于测试 `readTickerFile` ，遵循单一职责原则。

在TDD（Test Driven Development）过程中，使用Stub可以帮助我们在后续程序还没实现的情况下，先验证和完成目标功能。

Fake

Fake是用一个假的实现，来代替实际目标程序，该实现只适用于测试环境而不能用于生产环境。

上例的 `onError` 即是Fake的，只用来验证返回的错误信息。

另外Stub后面的 `callFake` 也是相同用法，虽然Stub了，但是也提供一个Fake的实现，写了些简单的代码来顺便验证它接收到的参数是不是我们所期望的。

Sinon里另外提供fake方法,结合了Spy和Stub的概念。

```
var fake = sinon.fake.returns('42');
sinon.replace(console, 'log', fake);
console.log('apple pie');
// 42
```

Mock

Mock主要是验证程序行为有没有按照我们预定的路径发生，它也可以返回预设数据，同时可以对交互进行跟踪，如调用的次数，调用的顺序。

比如 `processTickers` 的Test Case验证了 `'processTickers should call getPrice for each ticker symbol'` 。

数组参数的每个元素都会作为 `getPrice` 的参数调用一次。如果任何一个预期没有发生，测试会失败。

Spy

Spy实际上是通过代理来获取目标的被调用情况，如被调用了几次，使用什么参数。

Spy可以与真实的服务进行交互，对交互进行验证或部分模拟。

上面Spy的Test Case是参照Mock的例子写出来的，同样验证了数组参数被 `getPrice` 调用的情形，而且是按顺序调用。

参考

https://sinonjs.org/

npm run传递参数 ===

karma.conf.js使用以下设置时，unit test会不停地跑。

```
autoWatch: false,
singleRun: true,
```

想只跑一次，发现命令行参数不起效果。

```
npm run ng test --watch=false --code-coverage
```

原来需要加 `--` ，这是npm传递参数给script的方法。

```
npm run ng test -- --watch=false --code-coverage
```

`npm help run` 查看详细说明。

> As of npm@2.0.0, you can use custom arguments when executing scripts. The special option -- is used by getopt to delimit the end of the options. npm will pass all the arguments after the -- directly to your script:
>
> npm run test -- --grep="pattern"
>
> The arguments will only be passed to the script specified after npm run and not to any pre or post script.

不过不应该整天打这么长的命令行，复杂的命令都应该放到script里。

package.json

```
"test": "ng test --watch=false --code-coverage",
```

然后 `npm test` 就可以。

使用Mocha、Istanbul和Chai实现TypeScript单元测试和覆盖率

本文主要描述使用Mocha、Istanbul和Chai实现TypeScript单元测试和覆盖率的方法和过程，并简单记录SonarQube的相关配置。

Github: https://github.com/prufeng/tsexpress

关于JavaScript的单元测试和覆盖率，可参考：使用Mocha和Istanbul实现Node.js单元测试和覆盖率

Install

安装相关module和types。
Mocha是基本单元测试工具。
nyc用来统计代码覆盖率。
Chai则主要用来测试HTTP request。

```
npm i -D mocha
npm i -D nyc
npm i -D chai
npm i -D chai-http
npm i -D source-map-support

npm i -D @types/mocha
npm i -D @types/chai
npm i -D @types/chai-http
```

Mocha

mocha.opts

创建test/mocha.opts，配置如下。

```
--require ts-node/register
--require source-map-support/register
--recursive
--full-trace
--bail
test/**/*.spec.ts
```

app.spec.ts

创建test/app.spec.ts，添加以下test case测试当前的几个RESTful API。

```
import chai from 'chai';
import chaiHttp from 'chai-http';
import server from '../src/server';

let should = chai.should();
chai.use(chaiHttp);

describe('App Unit Test', () => {
    it('should get status 200', (done) => {
        chai.request(server)
            .get('/')
            .end((err, res) => {
                res.should.have.status(200);
                done();
            });
    });
    it('should get user list', (done) => {
        chai.request(server)
            .get('/users')
            .end((err, res) => {
                res.should.have.status(200);
                res.text.should.eql('respond with the user list
                done();
            });
    });
    it('should get status 404', (done) => {
        chai.request(server)
            .get('/wrongUrl2018')
            .end((err, res) => {
                res.should.have.status(404);
                // res.text.should.eql('respond with the user li
                done();
            });
    });
});
```

Unit Test

运行mocha命令，测试通过的结果如下。

```
mocha

  App Unit Test
GET / 200 13.293 ms - 198
    √ should get status 200
GET /users 200 0.662 ms - 31
    √ should get user list
GET /wrongUrl2018 404 7.099 ms - 1339
    √ should get status 404


  3 passing (55ms)
```

Istanbul(nyc)

package.json

修改package.json，增加nyc相关配置。
`include` 和 `extesion` 包含需要测试的source code， `reporter` 指明需要生成的report，缺省为text。

```
  "scripts": {
    "start": "npm run serve",
    "serve": "node dist/out-tsc/server",
    "test": "nyc mocha",
    "build":"tsc -p tsconfig.json"
  },
  "nyc": {
    "include": [
      "src/**/*.ts",
      "src/**/*.tsx"
    ],
    "exclude": [
      "**/*.d.ts"
    ],
    "extension": [
      ".ts",
      ".tsx"
    ],
    "require": [
      "ts-node/register"
    ],
    "reporter": [
      "text",
      "html"
    ],
    "sourceMap": true,
    "instrument": true,
    "all": true
  },
```

npm test

`npm test` 实际上是运行 `nyc mocha`，但直接运行 `nyc mocha` 得不到正确的结果，因为无法加载package.json里的nyc配置。
成功运行后参考结果如下。

```
--------------------|----------|----------|----------|----------
File                |  % Stmts | % Branch |  % Funcs |  % Lines
--------------------|----------|----------|----------|----------
All files           |    79.03 |    36.84 |    85.71 |    79.03
 src                |       74 |    33.33 |    66.67 |       74
  app.ts            |      100 |      100 |      100 |      100
  server.ts         |    56.67 |    33.33 |    66.67 |    56.67
 src/app/common     |      100 |       50 |      100 |      100
  errHandler.ts     |      100 |       50 |      100 |      100
 src/app/demo       |      100 |      100 |      100 |      100
  homeController.ts |      100 |      100 |      100 |      100
  userController.ts |      100 |      100 |      100 |      100
--------------------|----------|----------|----------|----------
```

如果生成了html report，可在coverage目录下打开index.html。html report的好处是可以点击查看每个文件的具体测试覆盖详情。

All files

76.36% Statements 42/55 **42.11%** Branches 8/19 **80%** Functions 4/5 **76.36%** Lines 42/55

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

File		Statements		Branches		Functions		Lines	
src		74%	37/50	33.33%	5/15	66.67%	2/3	74%	37/50
src/app/common		100%	4/4	75%	3/4	100%	1/1	100%	4/4
src/app/demo		100%	1/1	100%	0/0	100%	1/1	100%	1/1

SonarQube

Install SonarQube Plugin - SonarTS

https://docs.sonarqube.org/display/PLUG/SonarTS

sonar.properties

```
sonar.javascript.lcov.reportPaths=coverage/lcov.info
sonar.typescript.lcov.reportPaths=coverage/lcov.info
```

(Works without sonar.typescript.lcov.reportPaths configure from my test.
)

Reference

https://istanbul.js.org/docs/tutorials/typescript/

VSCode自动完成功能会在我们打左括号的时候自动完成右括号，可是，非常郁闷的是，在这个编辑状态下，却无法像其他编辑器一样通过按Tab或Enter键跳出右括号到行尾。
虽然VSCode有强大的按键自定义功能，但初学者看起来还是头晕。
而且居然百度不到，故起此贴。

在这个Issue下找到相对简单的解决方法，就是安装Tabout插件。
https://github.com/Microsoft/vscode/issues/22864

Github上还起了许多相关Issue，有心人可看看。

VSCode集成Git Bash作为Terminal

Windows下想用Git Bash作为VSCode Terminal，可尝试以下方法。

方法一

- Ctrl+Shift+P
- Select Default Shell
- Select Git Bash

默认为cmd，也可选powershell。

（我本机VSCode识别不到Git，Git Bash选项没出现。）

方法二

- Ctrl+Shift+P
- Open User Settings
- Change terminal.integrated.shell.windows

  ```
  "terminal.integrated.shell.windows": "C:\\windows\\System32\
  ```

  改为

  ```
  "terminal.integrated.shell.windows": "C:\\swdtools\\Git\\bin
  ```

进阶

加启动参数。

```
"terminal.integrated.shellArgs.windows": ["-l"],
```

e.g.
~目录下建立.bash_profile，输入以下内容。

```
alias gss='git status -s'
```

重启Terminal，即可输入gss查看效果。

我主要使用该功能来建立Git Command Alias。

参考

https://code.visualstudio.com/docs/editor/integrated-terminal

使用Mocha和Istanbul实现Node.js单元测试和覆盖率

基于Express，Mocha + Istanbul

1. Unit Test

1.1 Install Mocha

```
npm i -D mocha
```

1.2 Create test/test.js

```
var assert = require('assert');
describe('Array', function() {
  describe('#indexOf()', function() {
    it('should return -1 when the value is not present', functio
      assert.equal([1,2,3].indexOf(4), -1);
    });
  });
});
```

1.3 Run Test Case

Add `test` to package.json `scripts` node as below.

```
  "scripts": {
    "start": "node ./bin/www",
    "test": "mocha"
  }
```

Run `npm test` , will get result in the terminal.

```
> mocha



  Array
    #indexOf()
      √ should return -1 when the value is not present


  1 passing (5ms)
```

2. Code Coverage

2.1 Install Istanbul

新版本Istanbul命令行工具已经改为nyc。

```
$ npm i -D nyc
```

2.2 Run Test

修改package.json

```
{
  "scripts": {
    "test": "nyc mocha --timeout=3000"
  }
}
```

运行 `npm test` ，得结果如下：

```
> nyc mocha --timeout=3000



  Array
    #indexOf()
      √ should return -1 when the value is not present


  1 passing (8ms)

----------|----------|----------|----------|----------|---------
File      |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovere
----------|----------|----------|----------|----------|---------
All files |        0 |        0 |        0 |        0 |
----------|----------|----------|----------|----------|---------
```

因为没有测试到任何代码，所以上例各项覆盖率为0。

2.3 实例

添加如下代码供测试。

```
//calc.js
let calc = new Object();
calc.add = function(x, y){
    return x+y;
}
calc.minus = function(x,y){
    return x-y;
}
module.exports = calc;
```

新建calc.spec.js。

```
//test/calc.spec.js
var assert = require('assert');
var calc = require('../calc');

describe('Calculator', function() {
  describe('add', function() {
    it('add(1,2) should return 3', function() {
      assert.equal(calc.add(1,2),3);
    });
  });
});
```

运行测试，结果变为：

```
----------|----------|----------|----------|----------|---------
File      |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovere
----------|----------|----------|----------|----------|---------
All files |    83.33 |      100 |       50 |    83.33 |
 calc.js  |    83.33 |      100 |       50 |    83.33 |
----------|----------|----------|----------|----------|---------
```

修改calc.spec.js，添加测试用例。

```
//test/calc.spec.js
var assert = require('assert');
var calc = require('../calc');

describe('Calculator', function() {
  describe('add', function() {
    it('add(1,2) should return 3', function() {
      assert.equal(calc.add(1,2),3);
    });
  });

  describe('minus', function() {
    it('minus(1,2) should return -1', function() {
      assert.equal(calc.minus(1,2),-1);
    });
  });
});
```

运行测试，结果变为100%。

```
----------|----------|----------|----------|----------|---------
File      |  % Stmts | % Branch |  % Funcs |  % Lines | Uncovere
----------|----------|----------|----------|----------|---------
All files |      100 |      100 |      100 |      100 |
 calc.js  |      100 |      100 |      100 |      100 |
----------|----------|----------|----------|----------|---------
```

2.4 坑++

以上report还是有点问题，只包含了一个文件，并没有出现项目中的其他文件，说明Istanbul只是统计了测试中的文件。

```
var app = require('../app');
```

把App入口文件import进去test.js以后，得结果如下。

```
-----------------|----------|----------|----------|----------|--
File             |  % Stmts | % Branch |  % Funcs |  % Lines | U
-----------------|----------|----------|----------|----------|--
All files        |    82.93 |        0 |    33.33 |    82.93 |
 autotest        |    83.87 |        0 |       50 |    83.87 |
  app.js         |       80 |        0 |        0 |       80 |
  calc.js        |      100 |      100 |      100 |      100 |

 autotest/routes |       80 |      100 |        0 |       80 |
  index.js       |       80 |      100 |        0 |       80 |
  users.js       |       80 |      100 |        0 |       80 |
-----------------|----------|----------|----------|----------|--
```

要把所有文件包含进去，可以为nyc命令增加-a参数，即修改如下。

```
"test": "nyc -a --reporter=clover mocha --recursive"
```

Reference:

https://mochajs.org/
https://istanbul.js.org/

怎样安装NPM离线包

因为一些（你懂的）原因，工作环境无法直接使用 `npm install` 联网安装npm包，稍微花了点时间研究了一下，Mark下来，有机会看源码再补充。

最佳方案

别浪费时间了。
最好还是在网络环境下把所需的包全部安装好，再想办法搬回来。
尽管可以一个一个下载包，但是其中依赖关系却错综复杂，对于关系复杂的情况，逐个下载基本上不可行。

可行办法

假设没有依赖关系，则可以下载安装包，然后放到node_modules。注意文件夹名称一般要删掉版本号，才与package.json描述一致。

或者直接指向压缩包运行install命令。
`npm install <tarball file>`

这种方法对于有依赖关系的包不可行，就算已经把所有依赖包放到该安装包里的node_modules目录下，安装过程还是会去尝试连接到网络。

如下方法却可行，但却只是使用symlink方式连接到目标文件夹。
`npm install <folder>`

看了一下help，并不是bug，但总感觉不太友好。

```
Install the package in the directory as a symlink in the current
```

一定要npm install？

不是放到node_modules下就可以？为什么还要npm install？
除了更新package.json，刚好我用来试的是express-generator。如果只是放到/node_modules文件夹下，express命令行不可用。
npm install后，下出现express和express.cmd，命令可用。

所以对于这种要添加命令行工具的包，需要用npm install来安装，比如：exress-generator、TypeScript。

整合Bamboo和SonarQube实现Node.js自动化测试

1. Bamboo Node.js Plugin

1.1 安装Node.js插件

https://marketplace.atlassian.com/apps/1211241/node-js-support-for-bamboo?hosting=server&tab=overview

1.2 添加Node.js Executable Capability

否则在Build Plan里 `Add Task` 的时候，将会发现Node.js Task和npm Task没有Executable可选，需要Add New Executable。

注意：Node.js Executable的路径要配置到可执行文件名，即node.exe，npm不需要额外配置，Bamboo会根据Node.js配置得到npm.cmd的正确路径。

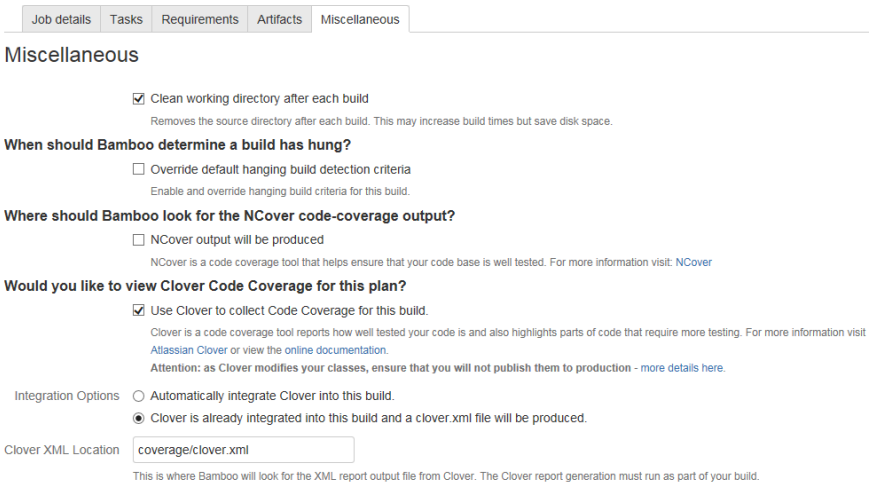
参考：
https://confluence.atlassian.com/bamboo/defining-a-new-executable-capability-289277164.html

2. Bamboo Plan

`Create a new Plan`，一个Job通常包括以下Task。

- Source Code Checkout
- Install Dependent Modules (npm install)
- Run Unit Test (npm test)
- Generate Lcov Report (coverage/lcov.info)
- Mocha Test Parser (parse mocha.json)
- Sonar Scanner
- *Parse Coverage Report (接收coverage/clover.xml)*

`Parse Coverage Report` 并不是一个Task，而是可以直接在Miscellaneous里配置。



`Run Unit Test` 用 `npm test` 而不是 `Mocha Test Runner`，主要是要整合Istanbul生成Coverage Report，详细参考下一节。

参考：
https://confluence.atlassian.com/bamboo/getting-started-with-node-js-and-bamboo-687213472.html
https://confluence.atlassian.com/bamboo/configuring-tasks-289277036.html

3. Bamboo+Mocha+Istanbul

程序部分可参考：使用Mocha和Istanbul实现Node.js单元测试和覆盖率

为了整合Bamboo和Mocha，需要以下插件。

```
npm i -D mocha-bamboo-reporter
```

修改package.json Scripts。

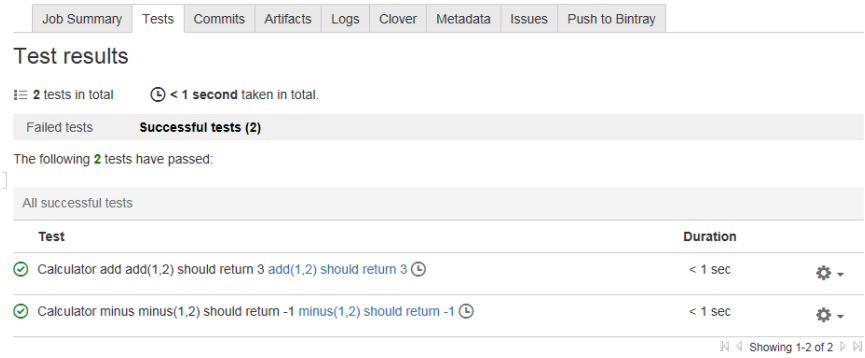
```
"test": "nyc -a --reporter=clover mocha --recursive --reporter m
```

mocha-bamboo-reporter用来生成mocha.json
`clover` 用来生成coverage/clover.xml。

4. Bamboo Run Plan

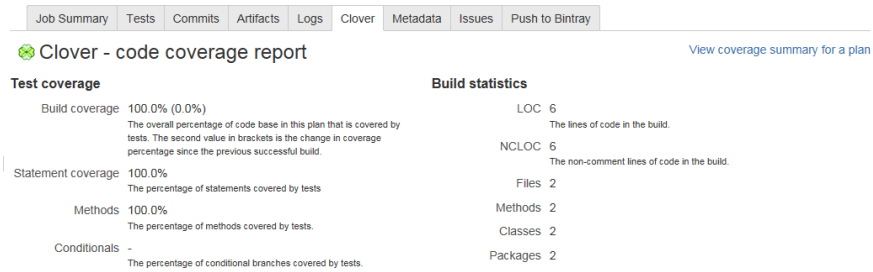
成功执行可得结果如下。

4.1 Test Result



4.2 Clover

点击 `View coverage summary for a plan` 可以看Summary和Historical charts。



5. Bamboo+SonarQube

5.1 Clover Not Support?

Bamboo和Node.js整合完成后，已经可以在Bamboo上看到Unit Test的结果和Code Coverage报告，但是Sonar Coverage始终为0，表明该结果还没有被Sonar读取。

```
//sonar-project.properties
sonar.core.codeCoveragePlugin=clover
sonar.clover.reportPath=coverage/clover.xml
```

以上配置尝试多次还是不行，Bamboo上也没看到什么有价值的log，只好从努力到放弃。（Any idea?)

5.2 Lcovonly

最后发现Sonar支持Lcovonly，成功配置如下：

```
//sonar-project.properties
sonar.coverage.exclusions=test/**/*
sonar.javascript.lcov.reportPaths=coverage/lcov.info
```

参考：
https://docs.sonarqube.org/display/PLUG/JavaScript+Coverage+Results+Import
https://docs.sonarqube.org/display/SONAR/Analysis+Parameters

当然要先生成lcov report，增加以下script。

```
//package.json
"lcov": "nyc report --reporter=lcovonly"
```

Bamboo上添加相应task生成lcov report，即运行以下命令。

```
npm run-script lcov
```

至于原先的Clover配置，还是保持了，因为Bamboo对Clover支持比较好，Build完之后可以直接在Bamboo上查看，它的Historical charts看起来也很直观。

ITIL - Information Technology Infastruture Library

ITSM - Information Technology Service Management

ITIL Life Cycle 5 stages: Strategy, Design, Transition, Operation, Continual Service Improvement

Service Strategy Process
4 Activities: Market Definition, Develop Offerings, Develop Strategic Assets, Prepare for Execution
Strategy Management, Service Portfolio Management, Demand Management, Financial Management, Business Relationship Management

Strategy Management
4Ps: Perspective, Position, Plan, Pattern
sub-process: Strategic Assessment, Strategy Generation, Strategy Execution

Service Portfolio Management
3 parts: Service Pipeline, Service Catalogue, Retired Service
SPM-Key Activities: Define, Analyze, Charter, Approve

Demand Management
Knew your customer
Predicting the Demand, Manage the Risk PBA: Patterns of Business Activities PBA attributes: Frequency, Volume, Location, Duration Value = Utility(Fit for Purpose) + Warranty(Fit for use) Service Package-Core Service/Supporting Service, Service Level Package 3 ways to deal with demand: Anticipating Demand, Influence Demand, Creating Demand

Financial Management for IT Service Profit = Price - Cost Key Activities: Accounting, Budgets, Charging(optional)

Business Relationship Management Ensuring Customer Satisfaction (like Deposit and Withdrawal)

sub-process: Maintain Customer Relationships Identify Service Requirements Sign up Customers to Standard Services Conduct Customer Satisfaction Survey Handle Customer Complaints Monitor Customer Complaints

Service Design Process Design Coordination, Service Level Management, Availability Management, Capacity Management, IT Service Continuity Management, Service Catalog Management, Supplier Management, Service Continuity Management, Information Security Management

Research, Develop Idea, Testing Service Design Package 5 Aspepts: STAMP: Solutions, Tools, Architectures, Metrics(KPI), Process 4Ps: People, Process, Product, Partner

Service Level Management Service Level Agreement: Service based, Customer based, Miti-level(Coperate, Customer, Service) Operation Level Agreement Underpinning Contract

Service Level Management 6 stages

1. SLA Preparation and Agreement
2. Service Level Monitoring
3. Service Level Reporting
4. Service Improvement Program Planning
5. SLA Maintenance

Defining SLA Baseline SLA Document the Service SLA Model(Reward only, Risk only)

Service Credit, Earn back Service Default

Availability Management 2 Techniques: Proactive, Reactive\ 2 Aspects: Service Availability, Component Availability 7 Rs: Redundancy, Reputation, Realiability, Reparability, Recoverability, Responsiveness, Robustness

AST: Agree Service Time DT: Down Time Uptime/MTBF: Mean Time Between Failure Downtime/MTTR: Mean Time To Repair

Capacity Management Process Capacity Planning Sever Capacity Planning Steps: Determine Service Level Requirement, Analyze current capacity, Plan for the future KPI:

Capacity Balance Act: Cost vs Capacity, sub-process: Service Capacity Management, Component Capacity Management, Business Capacity Management

IT Service Continuity Management Factors disrupt business DR: Dizarster Recovery ITSCM Life Cycle: Initiation, Requirement & Strategy, Implemetation, Operational Activities, BCP-Business Continuity Plan

Service Catalog Management Business Catalog, Technical Catalog

Service Supplier Management Supplier & Contract Database

Information Security Management CIA-Confidentiality, Integrity, Availability Break down into: policy, process, procedure, work instruction

Service Transition Process Transition Planning and Support

ransition Change Evaluation, Service Validation and Testing, Release and Deployment Management, Change Management

Operation Service Catalogue, Service, Request Fullfillment, Incident Management, Problem Management

Continual Design Improvement

Service Strategy Utility and Warranty Value Creation

Demand Management Financial Management for IT Service Business Relationship Management

Service Design Process Service Level Management Service Catalogue Management Service Availability Management ITIL best practice

目标

P>15%，L<15%

原则

- 尽量少亏，争取熊市大幅跑赢，牛市小幅跑输
- 价值为锚，业绩为核心，长期持有，耐心等待企业成长
- 业绩可能不及预期，但经营不能恶化，风险最重要
- 花时间研究公司经营好坏，低风险机会，而不去判断股价涨跌、经济形势
- 相信机会永远存在，耐心等待深度价值，审慎决策，从容买入卖出
- 坚持保持舒适仓位，既欢迎上涨又欢迎下跌

仓位

按照市场估值调整。

1. 权益类<=80%
2. 固收<=80%
3. 现金<=20%
4. 单个标的<=30%
5. 同类标的<=60%

选股

1. 扣非PE(TTM)<10
2. 连续3年ROE>10
3. 连续3年分红>5%
4. 连续3年业绩稳定增长
5. 行业龙头，品牌佳，毛利高
6. 管理层稳定，口碑佳，无损害股东利益记录
7. 负债比例行业平均范围内，融资利率低
8. 现金流稳定，匹配经营投入产出

交易

1. PE<5可分3档买入，如PE为5，4，3时分别买入20%，30%，50%
2. PE>5时必须分5档买入，如PE为10，9，8，7，6时分别买入10%，15%，20%，25%，30%
3. 换股遵循以上买入策略，根据买入分档卖出
4. 股票个数不超过20个

5. 卖出主要分以下几种情况
6. 足利卖出（如盈利30%－50%，卖出<20%）
7. 高估卖出（盈利>100%，股息<3%，卖出>50%)
8. 疯狂卖出（盈利>200%，市场极度乐观，形势走坏，最多保留10%底舱）
9. 仓位过高卖出（最低档盈利10%以上可取<10%仓位卖出，卖出后不回落到相应或更低价值档位不加仓）

商业银行主要监管指标

资本充足率

资本充足率 = 资本净额/风险加权资产总额

监管要求

银行	核心一级资本充足率	一级资本充足率	资本充足率
商业银行	7.5%	8.5%	10.5%
系统重要性银行	8.5%	9.5%	11.5%

核心一级资本充足率、一级资本充足率、资本充足率分别不得低于5%、6%、8%。

储备资本要求为风险加权资产的2.5%，由核心一级资本来满足。

国内系统重要性银行附加资本要求为风险加权资产的1%，由核心一级资本满足。

补充途径

- 核心一级资本：ＩＰＯ、定增、配股、可转债
- 其他一级资本：优先股、永续债、无固定期限资本债
- 二级资本：次级债、混合资本工具、二级资本债

存款准备金率

截至2019年1月15日， 大型金融机构：14%， 中小型金融机构：12%。

杠杆率

杠杆率=一级资本净额/调整后的表内外资产余额

商业银行并表和未并表的杠杆率均不得低于4%。

不良贷款率

不良贷款率=（次级类贷款+可疑类贷款+损失类贷款）/各项贷款×100%
=贷款拨备率/拨备覆盖率×100%

评估银行贷款质量，采用以风险为基础的分类方法（简称贷款风险分类法），即把贷款分为正常、关注、次级、可疑和损失五类；后三类合称为不良贷款。

五类贷款的定义分别为：

正常：借款人能够履行合同，没有足够理由怀疑贷款本息不能按时足额偿还。
关注：尽管借款人目前有能力偿还贷款本息，但存在一些可能对偿还产生不利影响的因素。
次级：借款人的还款能力出现明显问题，完全依靠其正常营业收入无法足额偿还贷款本息，即使执行担保，也可能会造成一定损失。
可疑：借款人无法足额偿还贷款本息，即使执行担保，也肯定要造成较大损失。
损失：在采取所有可能的措施或一切必要的法律程序之后，本息仍然无法收回，或只能收回极少部分。

拨备覆盖率

拨备覆盖率=(一般准备+专项准备+特种准备)/(次级类贷款+可疑类贷款+损失类贷款)×100%

拨贷比

贷款拨备率=贷款损失准备金余额/各项贷款余额 x 100%

要求

拨备覆盖率监管要求由150%调整为120%~150%，贷款拨备率监管要求由2.5%调整为1.5%~2.5%

银行可参照以下比例按季计提专项准备：对于关注类贷款，计提比例为2%；对于次级类贷款，计提比例为25%；对于可疑类贷款，计提比例为50%；对于损失类贷款，计提比例为100%。其中，次级和可疑类贷款的损失准备，计提比例可以上下浮动20%。

低于150%拨备覆盖率的超额贷款损失准备计入二级资本的数量不得超过信用风险加权资产的0.6%；高于150%拨备覆盖率的超额贷款损失准备可全部计入二级资本。

信贷资产根据金融监管部门的有关规定进行风险分类，标准风险系数暂定为：正常类1.5%，关注类3%，次级类30%，可疑类60%，损失类100%。

参考

《商业银行资本管理办法（试行）》
《商业银行贷款损失准备管理办法》
《关于调整商业银行贷款损失准备监管要求的通知》(银监发【2018】7号)
《贷款风险分类指导原则》
《商业银行杠杆率管理办法（修订）》

指定Python安装包下载地址

有时安装包从pypi.org下载不了，或者网络限制或使用内部仓库。

Pip Configure.

Windows: ~\pip\pip.ini

Linux: ~.pip\pip.conf

```
[global]
index-url= http://mirrors.aliyun.com/pypi/simple/
extra-index-url= https://pypi.org/simple

[install]
trusted-host=mirrors.aliyun.com
```

Pip Install

命令行里再指定也可以。

也支持本地安装。不过，这是个痛苦的过程，就像Maven、Npm等其他依赖管理工具一样。

```
pip install --help


Package Index Options:
  -i, --index-url <url>       Base URL of Python Package Index (
                              https://pypi.org/simple). This sho
                              to a repository compliant with PEP
                              laid out in the same format.
  --extra-index-url <url>     Extra URLs of package indexes to u
                              rules as --index-url.
  --no-index                  Ignore package index (only looking
  -f, --find-links <url>      If a url or path to an html file,
                              file:// url that's a directory, th
```

最受欢迎数据库排行榜

345 systems in ranking, March 2019

Rank			DBMS	Database Model	Score		
Mar 2019	Feb 2019	Mar 2018			Mar 2019	Feb 2019	Mar 2018
1.	1.	1.	Oracle	Relational, Multi-model	1279.14	+15.12	-10.47
2.	2.	2.	MySQL	Relational, Multi-model	1198.25	+30.96	-30.62
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1047.85	+7.79	-56.94
4.	4.	4.	PostgreSQL	Relational, Multi-model	469.81	-3.75	+70.46
5.	5.	5.	MongoDB	Document	401.34	+6.24	+60.82
6.	6.	6.	IBM Db2	Relational, Multi-model	177.20	-2.23	-9.47
7.	↑ 9.	7.	Microsoft Access	Relational	146.20	+2.18	+14.26
8.	↓ 7.	8.	Redis	Key-value, Multi-model	146.12	-3.32	+14.90
9.	↓ 8.	9.	Elasticsearch	Search engine, Multi-model	142.79	-2.46	+14.25
10.	10.	↑ 11.	SQLite	Relational	124.87	-1.29	+10.06
11.	11.	↓ 10.	Cassandra	Wide column	122.80	-0.58	-0.69
12.	12.	↑ 15.	MariaDB	Relational, Multi-model	84.31	+0.89	+21.21
13.	13.	13.	Splunk	Search engine	83.10	+0.29	+17.44
14.	14.	↓ 12.	Teradata	Relational	75.22	-0.75	+2.76
15.	15.	↑ 18.	Hive	Relational	73.00	+0.71	+16.00
16.	16.	↓ 14.	Solr	Search engine	60.01	-0.95	-4.80
17.	17.	17.	HBase	Wide column	58.80	-1.48	-2.14
18.	18.	↑ 19.	FileMaker	Relational	58.13	+0.34	+3.00
19.	↑ 20.	↓ 16.	SAP Adaptive Server	Relational	56.03	+0.29	-6.58
20.	↓ 19.	20.	SAP HANA	Relational, Multi-model	55.51	-1.03	+6.99
21.	21.	21.	Amazon DynamoDB	Multi-model	54.49	-0.45	+12.04
22.	22.	22.	Neo4j	Graph	48.58	+0.72	+7.68
23.	23.	23.	Couchbase	Document	33.80	-1.78	+0.90
24.	24.	24.	Memcached	Key-value	28.73	-0.72	-2.62
25.	25.	↑ 26.	Microsoft Azure SQL Database	Relational, Multi-model	27.93	+0.81	+3.31
26.	26.	↓ 25.	Informix	Relational, Multi-model	26.90	+0.54	-0.61
27.	27.	↑ 31.	Microsoft Azure Cosmos DB	Multi-model	24.83	-0.03	+8.07
28.	28.	↓ 27.	Vertica	Relational, Multi-model	22.07	-0.74	+1.77
29.	29.	↑ 32.	Amazon Redshift	Relational	20.89	-0.10	+6.70
30.	↑ 32.	↓ 29.	Firebird	Relational	20.17	+0.82	+2.14

https://db-engines.com/en/ranking

主要评价维度：

- 网站提及次数(Google, Bing, Yandex)
- Google Trend
- 技术论坛(Stack Overflow, DBA Stack Exchange)
- 招聘(Indeed, Simply Hired)
- 简历(LinkIn, Upwork)
- 社交网络(Twitter)

笔记是为了解决问题 笔记是为了取得成果 笔记不是为了记录过去，而是为了创建更好的未来 动手做笔记利用了人的身体记忆 笔记将解决问题的过程可视化

解决问题的基本流程 设定问题和Issue 整理课题并分解其结构 搜寻现场的信息 建立解决方案的假设 验证假设 决定解决方案 实行解决方案

将笔记分类 发现真正问题的笔记（搜集信息，问题在哪里） 建立假设的笔记（信息分组，事实->解释->行动，故事线剧本） 验证假设的笔记（准备提问列表,Where，Why，How） 取得成果的笔记（概括能力，加重号形式，金字塔结构，3的乘方法则，一个图表、一条信息）

以始为终 除了传达的内容必须具有价值以外，还需要让对方能够容易接受

将想法整理成目录 随时准备与他人共享信息和思考方法 区分事实和意见，基于事实寻找解决方法 PTI（Person，Timing，Interest）：向什么人，在什么时候，用什么方式传达最有效 无法具体看到事物，就无法得到真正的改善 3种图解：分组归类（grouping），矩阵（matrix），商业模型（business system） 通过3C（Company自己，Customer目标，Competitor对手）视点来提高客观性

真正的信息，是自己亲临现场调查获得的“一手信息”。 全球化思考：与拥有不同文化背景的人共享信息，要注重“事实的积累”。

笔记的终极目标是“提高自己” 保持思路的清晰和流畅，将自身价值最大化 思考训练 探求笔记：应该如何发挥自己的经验和能力，去做一些事情 整理内心的笔记 自省笔记术

人类虽然善良但却懒惰 需要一个能够让自己行动起来的机制

2018年书单

2018年开始记录，以后要尽量抽多点时间来读书。

这是2018年的书单，主要是下半年，上半年读书较少，也并没有记下来。

In Progress

- 《原则》——瑞•达利欧
- 《JavaScript测试驱动开发》—— Venkat Subramanian 毛姝雯译
- 《Node.js In Action》—— Mike Cantelon
- 《黄明哲正解道德经》——（这个是随便听听）

Done

- 《微服务：从设计到部署》——向阳 译
- 《大型网站系统与Java中间件实践》——曾宪杰
- 《架构探险：从零开始写分布式服务框架》——李业兵
- 《高性能服务系统构建与实战》——银文杰
- 《持续交付：发布可靠软件的系统方法》——Jez Humble, David Farley 乔梁 译
- 《用户故事地图》——Jeff Patton
- 《每天读点中国哲学》——胡适
- 《胡适的北大哲学课（卷一）》——（跟上面那本基本一样，对比查漏）
- 《你自以为的极限，只是别人的起点》——特立独行的猫
- 《公司理财（第3版）》——蒋屏
- 《明朝那些事儿（漫画版）（合集）》——当年明月著 狐周周绘
- 《宽容（中小学生必读丛书）》——房龙
- 《李鸿章传》——梁启超
- 《麦肯锡笔记思考法》——大岛详誉
- 《道德经》——老子 李若水评译

Quit

以下几本读到中间停了下来，以后有机会补回来。

- 《Spring Cloud与微服务实践》——翟永超

- 《大型分布式网站架构与实践》——陈康贤
- 《Java 8 In Action》—— Raoul-Gabriel Urma
- 《Node.js区块链开发》——朱志文

以下这本版本较旧，随便翻翻，但太啰嗦了，看不下去。

- 《JavaScript权威指南》——陆凌牛

2019年书单

突然怀疑我的肩周炎，应该跟看书有关。
书，看得不多。
可是，却看得极慢。
主要是地铁和睡前。
地铁挤得累。
晚上睡得早，半夜便会醒来。， 据说老年人都这样……

In Progress

- 《白帽子讲Web安全》——吴翰清
- 《The GoF Design Patterns Reference》——w3sDesign
- 《Node.js In Action》——Mike Cantelon
- 《Elasticsearch：权威指南》——Shay Banon
- 《顶级强者》——一杯八宝茶 （C992）
- 《叶凡秋沐橙》——（C509）
- 《未来简史》——尤瓦尔•赫拉利
- 《Python机器学习与量化投资》——何海群
- 《万物发明指南》——瑞安•诺思
- 《影响力（经典版）》——Robert B. Cialdini著 闾佳 译

Done

- 《JavaScript测试驱动开发》——Venkat Subramanian 毛姝雯译
- 《原则》——Ray Dalio
- 《揭秘：物联网原理、实践与解决方案》——周晨光
- 《失控》——沙法丽•萨巴瑞
- 《生活遇上大数据》——钱淑芳
- 《Angular TUTORIAL》——angular.io
- 《巴菲特之道》——Robert Hagstrom 著，杨天南 译
- 《黄明哲正解〈道德经〉》——黄明哲（听书）
- 《情绪急救：治疗失败、拒绝、内疚等因素导致的各种日常精神伤害的实用策略》——Guy Winch
- 《股市真规则》——帕特•多尔西
- 《DevOps Fundamentals》——Web Age Solutions Inc
- 《可转债投资魔法书》——安道全
- 《三国演义》——东方二次元•天津神界漫画
- 《国学诵·中华传统文化经典读本:大学·中庸》——刘敬东
- 《论语》——陈典 译注
- 《富爸爸财富自由之路》——罗伯特·清崎 莎伦·莱希特

- 《富爸爸穷爸爸》——罗伯特·清崎
- 《公司治理与内部审计》——郭涛敏 罗萍
- 《巴菲特致股东的信》——劳伦斯A•坎宁安 杨天南译
- 《麻省理工大学公开课：MBA金融学》——Andrew Lo（视频）
- 《公司理财师专业能力》——陈玉罡 王大中 史欣向
- 《数据模型记分卡》——斯蒂夫•霍伯曼
- 《道德经的奥秘》——曾仕强（听书）
- 《父母的觉醒》——沙法丽•萨巴瑞
- 《汉字有故事：亲近动物篇》——李海生
- 《资管不可能这么有趣》——方铭辉
- 《定投十年财务自由》——银行螺丝钉
- 《阴妻艳魂》——今何夕
- 《被讨厌的勇气》——岸见一郎 古贺史健
- 《聪明人是怎样掌控情绪的》——Joseph Jastrow著 张雪帆 译

Quit

- 《金融工程案例》——王晋忠
- 《修真聊天群》——圣骑士的传说（C435）
- 《大数据架构详解：从数据获取到深度学习》——朱洁 罗华霖
- 《乱世婚宠：夫人要逃婚》——木易肖潇（C768）
- 《重生都市仙尊》——洛书 （C1295）
- 《HTML5 Tutorial》——w3school
- 《Angular Fundamentals》——angular.io

书评

《被讨厌的勇气》

第一次看到一本书，觉得仿佛是自己写的。
相似的话，已对不同的人说过再三。
人生的每一个瞬间，都是自己选择的结果，别人怎么想怎么说，那是别人的事情。

《资管不可能这么有趣》

这本书主要不是讲产品，而是讲运营，也并不怎么有趣。不过确实讲得足够简单平实。

《父母的觉醒》

当孩子降临，我们有幸成为父母，总忍不住憧憬孩子以后会是什么样子。但一切于我们也是新的，或许我们应该多想想，我们的孩子到底需要什么样的父母？
这是一本好书，可是却读得特别慢。或许说理的内容偏多了，一直无法集中，脑子里乱糟糟的。相比之下，作者的另一本书要好看得多。

《巴菲特致股东的信》

有空去读读完整年报。
坊间传闻，多是拾人牙慧。
智者的言论，每一句都可能受用终身。
快乐投资，享受生活，活得很久……

《富爸爸穷爸爸》

盛名之下，必有可取之处。这是一本极好的财商教程，主要讲述对待财富的应有态度，以及积累财富的一些主要策略。

以前我就常常跟家人说，知道我们为什么穷吗？因为我们从来都没有想过钱！或者说不够贪钱。放弃一个又一个明显的赚钱机会，终致后来清苦。耕读之家，耻于谈钱，这大概是市场经济社会最悲哀的故事了。君子爱财，取之有道。爱财，并不有违忠恕之道。想要变富，须时时处处想钱，思考钱从哪里来，到哪里去，并采取行动。如此，方能保障一家老小过上体面的生活。

《富爸爸财富自由之路》

能带来现金流的才是真正的资产，财富来自于资产的积累。为了积累资产，成为企业主和投资者比雇员和自由职业者要容易得多。不过前者需要改变观念和思维方式，需要提高情商、财商等更多学校和书本以外的学习，因而显得更困难，被视为不稳定、不安全。然而波动并不是风险，风险也并不是危险。这些经验的积累，需要积极的失败和真切的损失经历作为养分。

《论语》

《论语》讲日常起居，读起来果然比《道德经》有趣得多。
《道德经》讲的是道和德，《论语》主要讲的是行。
读完终于知道孔老夫子为什么是最伟大的教育家。
知易行难。口若悬河，拾人牙慧，夸夸其谈，很多人都会，可是理解并做到极致，才能真正成之为圣人。

《三国演义》

画得还不错。

小时看三国，
只喜欢青梅煮酒，
三英战吕布。
现在看三国，
却感怀家国兴亡，
将军迟暮。

《可转债投资魔法书》

周末一口气看完。
有些啰嗦，有些小幽默，反反复复说的其实是一个意思——可转债是个好投资。
不过现实没那么美好，已经踩过一些坑之后再来看，感觉还是有些不一样。风险部分还是可以再挖掘一下。
投资，说到底都是风险和时间的游戏。

《股市真规则（畅销版）》

虽然本书的题目为《The Five Rules For Successful Stock Investing》，但却是难得的一本研究价值投资方法论的书籍。内容包括原则，应当避免的错误，竞争优势，财务报表基础和陷阱，公司分析，估值，以及行业分析。不同行业公司特点的分析占了比较大的篇幅，别具特色，值得偶尔翻翻。
成功投资的五项原则为：
1.做好你的功课（基本面分析）；
2.寻找具有强大竞争优势的公司；
3.拥有安全边际（估值）；
4.长期持有；
5.知道何时卖出。

《情绪急救》

对有用的人来说，这或许是非常有价值的一本书。
我们的精神和身体一样，都可能会生病。在人生某些关键时刻，承认自己存在感情和心理的困扰，得到适当的治（自）疗，我们才能更坦然地走下去。
生活在记忆的感觉之中，是非常痛苦的事情，走出来，从过去的经历中汲取营养，理性地看待自己，才能真正感受到，当下的幸福。

《原则》

读后感写成了一篇长文——《世俗的成功法则》。
公众号：星之空（rufengp）

2020年书单

In Progress

《白帽子讲Web安全》—— 吴翰清 《The GoF Design Patterns Reference》—— w3sDesign

《Node.js In Action》—— Mike Cantelon, Marc Harter, T.J. Holowaychuk, Nathan Rajlich

Mike Cantelon 《Node.js in Action, SecondEdition》——Alex Young, Bradley Meck, and Mike Cantelon 《Elasticsearch：权威指南》——Shay Banon

《顶级强者》—— 一杯八宝茶 （C1184） 《一世豪婿》—— 发飙的天空（C874）

《未来简史》—— 尤瓦尔•赫拉利

《万物发明指南》—— 瑞安•诺思

《目标管理法》——陈伟

《福尔摩斯探案集》——阿瑟•柯南•道尔 著 李嘉 译

Done

- 《孟子》——蔡志忠
- 《影响力（经典版）》—— Robert B. Cialdini著 闾佳 译 《论语别裁》——炜爵爷（听书，比较啰嗦） 《迷藏》——亦静 《百度SEO一本通》——潘坚 李迅 《自组织：互联网+企业管理创新》——方永飞 《合伙制：互联网时代的高效企业组织模式》——李振勇 《手把手教你开公司》——张俊锋 《穷查理宝典：查理·芒格智慧箴言录》——彼得·考夫曼 著 李继宏 译 《次第花开（2017修订新版）》——希阿荣博 《海底两万里》——儒勒•凡尔纳 著 聂尊阳 编 《一平米健身：硬派健身》——斌卡 《十万个为什么（伊林经典科普小丛书）》——[苏联]伊林 著 董纯才 译

Quit

- 《Python机器学习与量化投资》——何海群
- 《都市风云》——易克（C712）

书评

《十万个为什么（伊林经典科普小丛书）》——[苏联]伊林 著 董纯才 译

和小孩一起看的，从屋子里熟悉的东西问起，足够吸引人。不过看得出来原著是一本比较老旧的书，简笔画的插图，讲的都是碗、茶壶之类简单原始的家具。

《一平米健身：硬派健身》——斌卡

挺好的，连我这种不锻炼的人，也翻着翻着看完了，还下了单买弹力带。 当然，我知道自己是一时兴起，所以选的是拼多多畅销款。 不过这本书的真正价值在于，它不只告诉我们怎么样，还详细说了为什么，每一章后面长长的英文引用，表明了作者的科学态度和良苦用心，是我喜欢的类型。

《都市风云》——易克（C712）

虽然很想知道乔梁最终有没有大仇得报，但还是决定不跟了。 看过前面，已经满足了对市府大院的大部分想象。 其他的，无非就是政治，阴谋，钱色交易。

说实在的，不是消遣，很难再看下去。 不过也打开了一扇窗，提示可以多找点不同类型的书看看。

《海底两万里》——儒勒•凡尔纳 著 聂尊阳 编

就像看了一系列的海底世界纪录片，然而这居然是1870年的作品！钦佩之情，无以言表。 其中写到的海洋生物，或许就有几百种。 更不用说鹦鹉螺号潜艇的设计细节，比如电力的使用，氧气的压缩，海底漫步的潜水服…… 就算放到今天，仍然属于高科技，而在当时，远远还没有发明出来！

《次第花开（2017修订新版）》——希阿荣博

偶然打开，便一口气读完，许是与佛有缘。 虽非信众，佛里的智慧，也足以引人开悟。 愿众生，皆能如佛陀所愿，离苦得乐。
——————— 世间的事就像儿童的游戏，可以一直玩下去，可是随时停下来，也就结束了。

造成痛苦的根本原因在于我执。

佛陀说，一切痛苦的根源在于我们长期以来对自身及外部世界根深蒂固的误解，执幻为实。

有时候，人们并非不快乐，只是以为自己不快乐而已。

如果我们认为生活中有痛苦是正常的，人生本来如此，我们就能更好地集中精力处理问题本身，而不是无谓地纠缠在愤愤不平的情绪中。这种情绪只会增加挫败感和怨气，却丝毫不能帮助我们富有建设性地应对生活的难题。

如果痛苦、尴尬在所难免，我们最好让自己有所准备。这样做的好处是，虽然该面对的问题还得面对，该经历的痛还得经历，我们却不再那样感到苦，不必再承受额外的恐惧和焦虑。

寂天菩萨曾说："问题若有办法解决，就不必担心；若没办法解决，担心也没有用。"

《穷查理宝典：查理·芒格智慧箴言录》——彼得·考夫曼 著 李继宏 译

如果早知道这是一本真正教人智慧的书，我不会等到现在才看。 直接叫《芒格演讲》还好，什么宝典箴言之类，我对这类题目多少有些偏见。 我常常好为人师地说，人要有点常识，其实我并不能给常识下一个精确的定义。芒格这本书很好地解决了这个问题。 所谓常识，即一个多学科的多元思维模型，能保证自己尽量客观地看到事实的全部，然后再做出判断。 拥有广泛的常识，且能总是保持客观而不受情绪和心理的影响，这就叫做智慧吧。

《手把手教你开公司》——张俊锋

内容较全面，既包括公司的选址注册，合伙规则的制定，股权分配和融资，也包括招聘，绩效，财务，和营销等公司管理实务。

多看看此类书籍，或许可以成功打消我们创业的念头。

《合伙制：互联网时代的高效企业组织模式》——李振勇

相当精炼的一本书，短短篇幅讲清楚了什么是合伙制，包括怎么找合伙人，怎样制定合伙规则和分配机制，如何开展股权激励和规避风险。

《自组织：互联网+企业管理创新》——方永飞

超级扁平化，精英小团队。 自组织，去中心化，开放，无边界。

不要和我比懒，我懒得和你比

是的，我承认，写这个题目，主要是为自己的懒进行辩解。

懒的觉知

在过去相当长的时间，我一直有一个误解，觉得所有人都跟我差不多。
默认别人也是这么想的，自然也得了不少教训。

后来沟通工作做得多了，才明白，人与人之间的差别，并不是我预先可以想象的。

甚至于家人，也是各有各的不同。
甚至于自己，现在和过去，也有太大的差距。

我一直都觉得自己是比较勤奋的，直到遇到大嘴妈。
很快就被贴上了懒的标签。

一时竟无法辩驳。

我书架上待看的书单便是证据，如：
《极简主义》，
《越简单，越美好》，
《无所事事的艺术》，
《闲暇处才是生活》……

懒的好处

要为自己辩解，首先肯定是要列举懒的好处。

比如，我们可以说懒是第一生产力。
懒得走路，所以才有了车，
懒得开车，于是又有自动驾驶，
更不用说什么洗衣机、洗碗机和扫地机器人了。

不过，这种论点显然站不住脚，
因为这只是一种现象的罗列，无法证明其因果关系，而且发明家看起来更多是相对勤奋的人。

问题并没有解决。

真正的问题是，
为什么你可以这么懒！

懒的本质

这涉及到更深层次的心理决策机制，即当我们面对一件事情的时候，我们为什么决定不做。

而且大多数时候，我们甚至都不知道自己做了决定。（变成懒癌了。）

观察生物的活动，本质上是为了满足生存的需要，当生存没有受到威胁时，都倾向于什么也不做。（吃饱就睡。）

所以，懒是天生的。

后天的教育基本上都在强化人的勤奋自律，似乎也佐证了这一点。

懒的反面

那有人说，我热爱工作，工作使我快乐呢？

高级需求本质上是低级需求的正反馈。

知道每次努力都可以得到你想要的东西，何乐而不为。

想要的东西也会随着反馈环的进化而有所不同，或兼而有之。

可能一开始是金钱、荣誉，后来是知识、人际关系，再后来是失败和挑战……

所以，优秀真的是一种习惯。

但目前就大多数人来说，驱使我们停不下来的，并不是对工作的热爱，而只是一种焦虑的情绪。

焦虑的根源还是需求得不到满足。

学习工作到很晚，说到底还是知识能力不够。

夜深了还不想睡，通常来说是一种补偿心理。

一整天都处在别人的支配和控制之中，好不容易有片刻完全自主，当然舍不得。

就像人们开车回到楼下，却坐在车里发呆或抽烟的感觉。

跨出一步，便要从一个喧嚣，进入另一个喧嚣。

懒的意义

并不是所有的外界反馈都是正面的，也并不是所有人都能从外界反馈获得正能量。

短暂的休息，片刻的自主，是我们找寻自我存在和自我价值的重要手段。

自我存在感和自我价值感是一切自主行动的力量源泉。

懒的本能是生命活动的周期性决定的。

懒，选择不做，实际上是在说，我现在就要得到休息，而不是延迟享受。

延迟享受是理智的投资，牺牲眼前利益以换取长期更大收获。

懒则促使人们停下来，享受当下，体验生命的存在。

另一种懒

如果一个本来勤奋的人，突然变得很懒什么也不想做，这又是为什么？

这可能是受到心理创伤之后的一种逃避。

停下来，是发现感受不到自己的存在，突然没有了方向。

因为缺乏存在感，总是觉得自己无能为力，可有可无。

所以极度害怕失败，宁愿躲在当下，不肯向前一步。

潜意识里认为只要不去行动，就不会有挫折，什么也不做，就不会出错。

过度脆弱和敏感，甚至连普通社交也变得困难。

其实人与人之间，并没有多么大的沟壑，向前一步，海阔天空。

就像A股，尽管萎靡多年，但是只要一根大阳线，三观尽改。（2019-02-25）

懒的故事

达芬奇以勤奋智慧出名，但传说他在画《最后的晚餐》时，却整天无所事事，迟迟也不开工。

教堂主事的院长极为嫉妒，便到公爵面前诋毁他，说他太懒不做事。

达芬奇不免要解释一番，并且说——
当我什么也不做的时候，才是我最高效的时候。

最后，又过了一段时间，达芬奇才开始动工，一气呵成，并且把院长作为了犹大的原型。

我觉得，我很可能也是这种人。
当我什么也不做的时候，才是我最充实的时候。

不要老骂孩子，打就行了

我经常跟大嘴妈说—— 不要老骂孩子，打就行了。 我是认真的。

每当大嘴妈辅导作业的时候， 仿佛远远就能听到机关枪的声音， 满屋子都散发着浓浓的火药味。

这个怎么做！ 哥哥一脸茫然地看着大嘴妈的脸， 好像不知道要干什么。 接着又是一顿吼。 循环往复。

我说，怎么只听见你在说？ 到底谁才是要做作业的那个？ 哥哥一直就那样呆呆地看着大嘴妈，小心翼翼，察言观色。 我觉得他根本就没心思在作业。

你行你上！

可是，当我在大厅或房间里给哥哥分析问题的时候，大嘴妈在厨房都忍不住插话，这道题应该怎样怎样…… 我说我问的是哥哥呀？ 哥哥一脸憋屈，默不作声。

虽然我的辅导结果遭质疑，辅导过程也曾被中途接管，但现阶段还是有必要继续这个工作。 因为我发觉哥哥的沟通还是有些问题，之前有段时间甚至跟大嘴妈一样，对人大喊大叫。 大嘴妈一般只会对他那样，他却对着每个人这样。

而且大嘴妈对妹妹和别家小孩，都轻声软语的，这样似乎也有些不公平。

其实之前也有建议大嘴妈少些骂哥哥。 大嘴妈说，我有骂人吗！ 我，笑而不语。

大嘴妈的父母对他们也多是这种沟通方式，现在，大嘴妈又照搬到了自己的孩子身上。 但不是每个人都可以适应，表现在哥哥身上，就是拒绝说话，死扛。

表现得好的话，他可能最终会弄明白大嘴妈想要的是什么，成为大嘴妈理想的样子。 表现得不好的话，他可能会觉得自己这也不行，那也不行，终将一事无成。 但无论哪种情况下，他都成不了他自己。

我小时候的情况好一些，完全没人管，只有在闯祸人家告上门的时候。父母肯定都是护着自己的小孩的，所以这时内心也获得了极大的支持。父母不怎么管束，可能会有些风险，但于我却成了好事。

父母少干涉，所以从小就习惯独立思考，别人的意见永远只是参考，开心或伤心的结果也习惯自己承受。 这样成长起来的，是有些倔强的内心力量。

相信只要是自己想，没有什么做不好的。 无论遇到什么挫折，也都视为自己注定的选择或际遇，纵使难受，也从不抱怨。

如果可以，我觉得这是我认为最值得传承的财富了。

可笑的是，我自己心里明白，唯有孩子自己经历了足够的困难，才能得到这个奖赏。 就像你跟一个沮丧的人说： 没事的，这只是小case。 他只会觉得你根本就不理解他。

所以，不要老骂孩子。

不要剥夺他们犯错的机会。 不怕犯错，才能乐于接受挑战，对生活保持好奇心，才能知道，每个关卡背后，总会有一些新的、有趣的东西。

不要妨碍他们认清这个世界的本来面目，以及，更重要的，认清楚他们自己。 这样，当他们面对人生的关键时刻，才能真正遵循自己的内心，做最真实的自己。 就算，当他们妥协的时候，也知道，这终究还是自己的选择，尽管不太理想，也能坦然面对。

这个喧嚣的世界，最难是卸下面具诚实面对自己。 当我们打孩子的时候，宣泄的也多是我们自己对自己的愤怒。

你居然敢顶嘴！ 潜台词是，你居然敢挑战我的权威。 但是我们又有什么权威呢？ 只不过是他们没能成为我们希望的样子。

而这期望背后，躲着我们儿时怯懦的身影。 我们真该和自己好好相处。

我们，只是平庸的父母。 我们的小孩，也只是平凡的小孩。

相对于经常骂，我是觉得不如打的。 不过如果打的标准过低，那也就跟骂没什么分别了。 我没有被父母打过，所以也没什么发言权。

但是这焦虑却极具传染性。 每当大嘴妈发脾气的时候，我也总是觉得火大，而且难以控制。 相比之下，就算股灾一天亏掉几个月工资，也丝毫不影响我晚上辅导作业的心情。 （嗯，是有些夸大了，但情绪波动的对比是真实的。）

这情绪必然也会传递给孩子。 然而孩子最终必须学会处理好自己的情绪，相信自己没那么差，才能做出真正理智的取舍。

我们都知道，孩子主要通过模仿来学习。 孩子也许并不太需要我们告诉他们怎么做，而只是缺少——幸福生活的父母。

我希望孩子永葆内心的自由自主，真正懂得尊重自己，尊重他人，知道在必要时去寻求帮助。

我希望孩子看到我的时候心是安的， 他们无需表现什么，成为什么，只需呆在我身边，玩玩具也好，哭闹也罢， 我都可以静静看着他们，满心欢喜和富足。

不要老骂孩子，打就行了（二）

——控制欲与负疚感

当我说不要老骂孩子，打就可以的时候，
其实我的意思是，犯不着这么生气。
不过这话肯定也是冒犯了。
我正在气头上，你跟我说没必要？
其实主要是，我觉得——
孩子生活中遇到的大多数事情，都是小事。
既然是小事，说说也就好了，何必大动肝火。

但很多人恐怕不能认同这个观点。
如果不是管得严，早就变成牛了！

对于一个孩子，我觉得，
很多时候，是我们要求过高了。
做不到，是一种正常现象。

大人做错了事情，第一时间都是惴惴不安，想办法逃避，补救，祈求原谅。
小孩子也一样，而且，会更快忘记。

我觉得，家长是过于焦虑了。
或者说，一直生活在焦虑之中。

我们就像是一个机器人，规定了几点钟要做什么，如果超过了预订的时间就会呱呱叫。

机器人说，我呱呱叫是因为你们一动不动。

事实上，孩子已经在行动了，只是有时候动作慢一些，笨拙一些，有时候有些心不在焉。

孩子的自制力是有限的。
在孩子的价值系统里，我们所以为重要的，往往排在许多其他事情之后，比如吃和玩……

我就知道你会这样！
你就是这种人！
你总是这样子！ ……

判决性的语言，让人无力反驳，
还可能会不知不觉建立抵抗的潜意识。
工作中我们会竭力避免说这种话，因为会令沟通无法进行下去，可是家庭里却常常充斥着这种暴力。

其实这背后的潜台词是——
你本来应该那样，但是……

这就是我们的控制欲，
而我们却浑然不觉。

事实上，我们对孩子远没有我们所自以为的那么了解。
假如我们多点听而不是说，可能还能知道得多一些。

老子说，天地不仁，以万物为刍狗。

天地对于万物是没有恩情的，这是天地的伟大之处，所以能做到生而不有，为而不恃，功成而弗居。

父母对子女也是没有恩情的，子女的生命并不是父母赐予的，只是自己自私行为的结果。
所以父母应该与子女平等相待，而不是以此要挟他们成为自己理想的延续。

反过来，孩子也不是天生就欠父母要来还债的。
他们有他们自己的使命，要去寻找他们自己的自然之道。

孩子不关心我们所追求的东西，是一种正常现象。
如果恰巧他们喜欢，我们应该满怀感恩。

可是，我为他们付出了全部。
我们给孩子最好的，买最好的品牌，报最好的培训班，甚至有人卖房给孩子请顶尖的艺术老师。

但是学习艺术的，又有多少最终成为了艺术家？

任何投资都是有风险的。
付出不一定有回报。
你有一定的几率会成功，并不代表你一定会成功。

连肥皂剧里的男（女）二号都懂得，
我爱你是我的事，你爱不爱我却是你的事。

有时候我们说放弃了许多，其实也不过是时间的选择。
即使当时不放弃，现在也未必就能怎么样。
只是这样挂嘴边，多少可以让我们自己感觉好受一些。

然而听者却从此背上了沉重的负疚感。
忠于自己，就辜负了父母的期望。
听父母的话，就偏离了自己的心。

所以父母还是要有自己的事业和追求，不要成为孩子的奴隶。
某天我们自己放弃了，也是我们自己的选择，与孩子无关。

许多人无法承受这责任，他们选择不生孩子，这是极明智的。

与不合理的付出对应的，往往是更不合理的索取。

怕接孩子迟到了，我们赶路太急摔跤。
等我们见到孩子，会发现这也不顺眼那也不顺眼。

都是因为你！

我们都有自己的情绪要宣泄，但是这真的与孩子无关。

超越自己的能力，为孩子提供好到虚幻的条件，是不现实的。

如果我们自己已经被压垮了，我们的孩子，该怎么站起来？

世俗的成功法则

> 人人都有长处和短处，人人都能在生活中扮演重要角色。大自然塑造万事万物皆有目的。你最需要的勇气不是驱使你战胜别人的勇气，而是不管其他人对你有何冀望，你始终坚持做最真实的自我的勇气。
>
> ——《原则》

读过的书中超预期的不多，《原则》算是其中一本。

想了一下，把题目定为《世俗的成功法则》，算是对过往所理解的成功学的小总结。

所谓成功，无非解决以下问题，亦即第一条原则。

> 独立思考并决定：（1）你想要什么；（2）事实是什么；（3）面对事实，你如何实现自己的愿望

你想要什么

许多人根本不知道自己想要什么。

我也不知道。

刚毕业的时候，有人问我目标。
我说，有一份稳定的工作，有老婆孩子，有周末可以逛的公园。

如今，一切近在咫尺。

是不是感觉已经到达了人生的巅峰？
然而并没有。

程序员，销售，主管，架构师，合伙人……
工作的部分，算是很努力地尝试了。

然而，只是一份工作。

儿子，父亲，丈夫，兄弟，朋友……
生活的部分，仍需努力。

目标必须是自己的

我们这一代人（中许多人）的悲哀在于——
没得选，
或约等于没得选。

这是一个剧变的时代，
我们的父母，却极少能给予有效的建议。
比如高考时，打工时，遇到挫折时……

很多时候，我们也根本不清楚即将面临的是什么，甚至无法与自己好好相处。

于是，只能选择责任。
对父辈艰辛的理解和现实生活的压力，更增加了我们的迂腐和愚忠。

连电视剧里也说——
不要大声责骂年轻人，他们会立刻辞职的，但是你可以往死里骂那些中年人，尤其是有房有车有娃的那些人。

但责任并不是目标。

> 不要混淆目标和欲望。

当我们的责任完成时，我们还能做什么？想做什么？

工作是为了养家糊口，却不仅仅是养家糊口。
结婚是家族的大事，却又仅仅是自己的事。

年龄渐长，我们又注定会看到我们最亲爱的人，一个一个离去，却无能为力。
人世间最大的悲痛莫过于此。

然而当他们离去时，我还是我吗？
我是不是突然变得什么也不是？

到了不惑的年龄，却依旧不能不惑。

不管怎么说，目标必须是我们自己的。
我是什么，要由我们自己来定义。

妈妈说，不要委屈自己。

目标必须是理想化的

或者说，必须赋予我们的目标一些理想化的意义。

> 不要把成功的装饰误认为成功本身。

锐•达利欧的人生目标是，有意义的工作和人际关系。

一个检验目标的好方法是，想象你的目标突然实现了，接下来该怎么办？

比如，突然实现了一个小目标，赚到了一个亿，该怎么办？

假如你的答案是继续赚钱，那赚钱本身应该可以作为目标。
（但要注意，钱本身是没有价值的，钱的价值在于它当下可以买到的东西。）

假如你的答案是各种玩，那似乎玩才是你的真正目标。
（此时应该专注于玩而不是那么拼命赚钱了。）

当然，也要适当地排下优先级。

> 尽管你几乎可以得到你想要的任何东西，但你不可能得到你想要的所有东西。

事实是什么

这个问题的回答，决定了一个人成就的高度。

选择比努力更重要。
我们的所有决策，都是基于我们所看到的事实。

掌握足够多真正客观的事实，才能帮助我们做出最优抉择。

> 最重要的事情并不是预知未来，而是知道在每一个时间点上如何针对可获得的信息做出合理的回应。

真相是什么

最近关于原生家庭的话题很火。
为什么要摆脱原生家庭的影响？
因为家庭生活里长期形成的某些根深蒂固的偏执，最终会使我们看不清世界的本来面目。

比如我爸就经常喜欢指导我们要怎样怎样做生意，但是事实上，在他的人生经历中，鲜有成功的经商经验。
假如按他说的去做，迟早要掉沟里。

我至今犹记得十几年前去拜访一个亲戚长辈的场景。
那时我哥刚跟人合伙开了个小店，本来挺开心的进门，谁知劈头就飞来一句:

“听说你自己出去开厂了？如果是真的，那就是疯了！”

竟无言以对。

我们的父母长辈，多有严重的小农意识和小市民心态。

比如，崇尚自给自足，羞于向别人求助；
小富即安，缺乏自律，害怕改变，懒得多向前一步；
自私自利，贪小便宜，做事缺乏长远考虑；
嫉妒心和虚荣心重，透支未来也忍不住要攀比……

想一想，长期在这样的环境熏陶下的我们，要多艰难才能长成自己喜欢的样子。

所以，每当我们面对现实，要时刻保持头脑极度开放、极度透明。
多问问自己，真相到底是什么？
我们身上可能有哪些偏见会阻碍我们认清事实？

客观规律是什么

这个世界上唯一不变的东西，就是变化本身。

当我们遇到新问题，最好先假设我是无知的，我的观点可能是错的。
Keep foolish，keep hungry。

停止幻想——事情应该怎么样怎么样。
多问——我怎么知道我是对的？

普遍的规律是什么？
有哪些科学的分析方法和系统框架？
成功的人和案例在哪里？

符合客观规律的，才可能是对的。
确保总是用可信度加权的方法来进行决策。

怎么办

一个人不可能两次踏进同一条河流，但是每一条河流都是相似的。

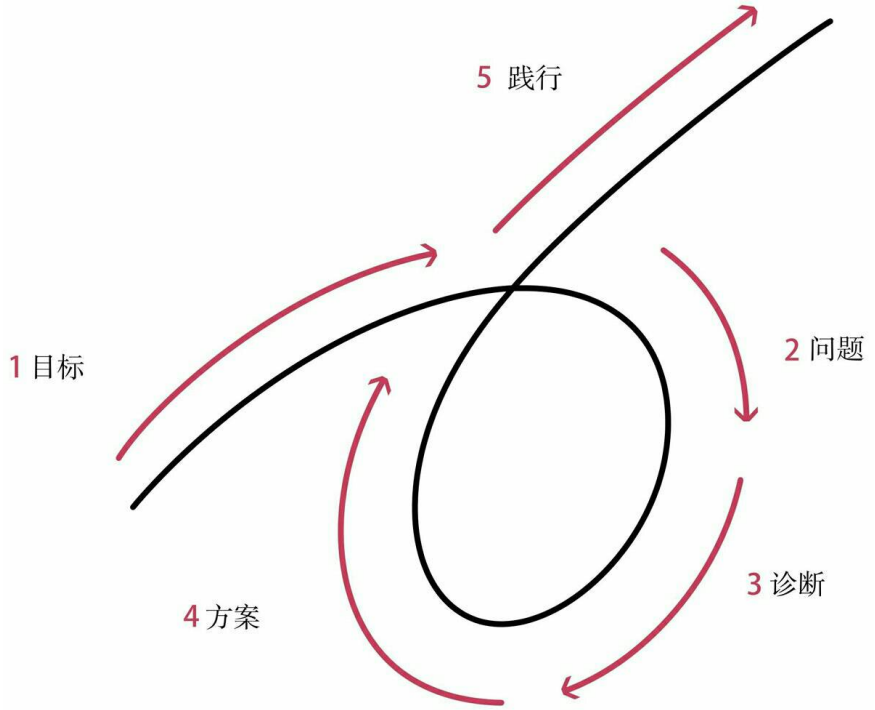
锐•达利欧提出五步流程的方法，来帮助我们实现螺旋向上的人生。

这其实并不是什么新理论。
它类似于质量管理的PDCA，即计划（Plan），执行（Do），检查（Check），处理（Action）。
也类似于软件工程里的迭代，一次次循环开发出更满意的产品。

五步流程

1. 有明确的目标。
2. 找到阻碍你实现这些目标的问题，并且不容忍问题。
3. 准确诊断问题，找到问题的根源。

4．规划可以解决问题的方案。
5．做一切必要的事来践行这些方案，实现成果。

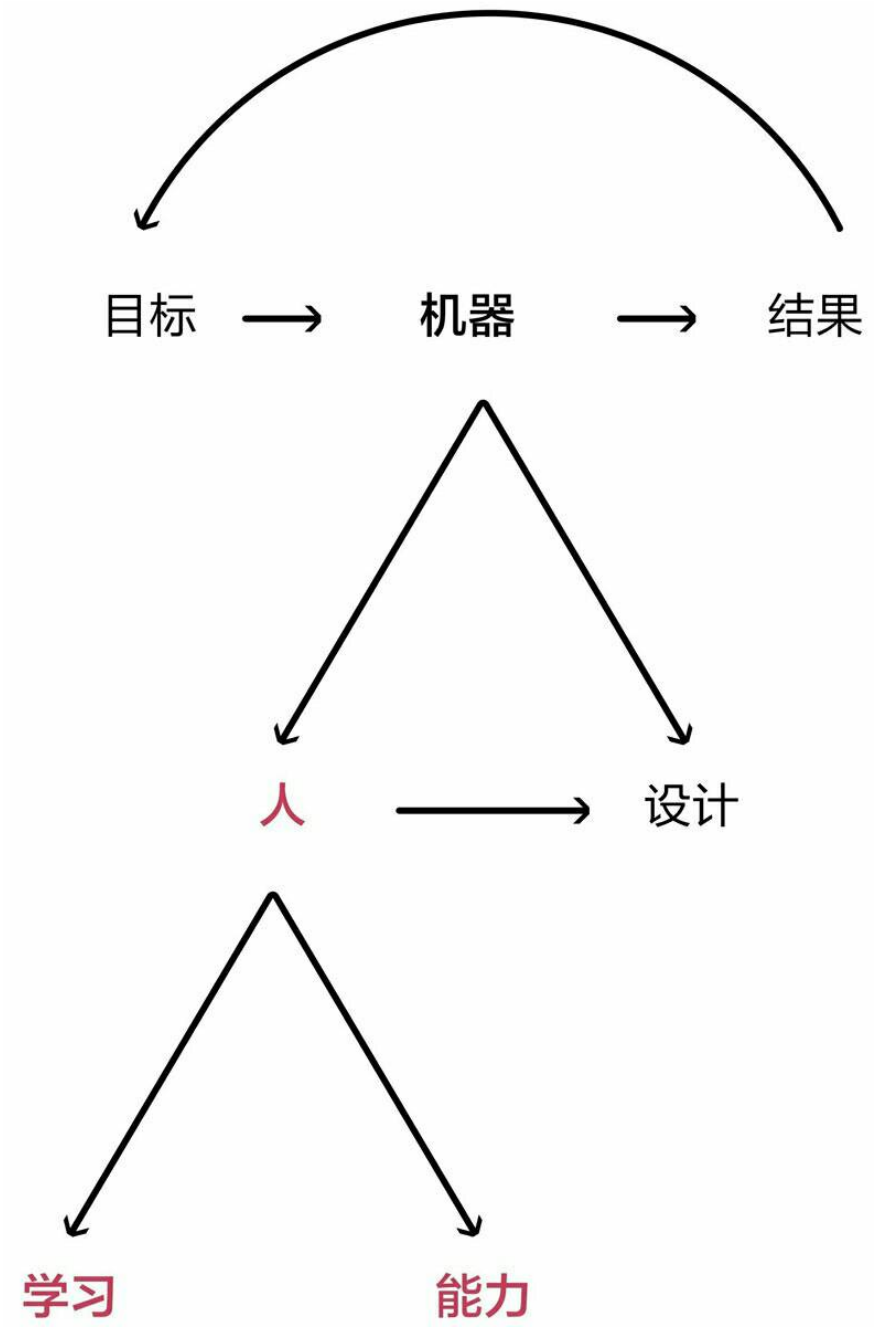


如何提高效率

五步流程简单易行，只需像机器一样执行。

把决策和执行的过程都变成可以清晰描述的算法，把所有事情系统化流程化和自动化。

简单来说，就是要按原则做事。



按原则做事

就像时间管理技术教给我们的，在固定的时间做特定的事情。
按原则做事，相当于提前想好，遇到什么情况做出何种反应。

生活中大多数的事情都是重复发生的，也就意味着，我们只需要很少的时间处理大多数事情，而可以把大多数时间花在比较难（有趣）的地方。

现在人们已经把许许多多重复的事情都交给了机器人。

不过身边为一件小事反复纠结的人还真不少。
明明几分钟就可以决定的事情，却犹犹豫豫，迁延时日，甚至夜不能寐。

应该仔细分析问题出在哪里。
如果缺少资料，就要大力搜集事实信息。
过于复杂则可以进一步分解，分别分析，再加权汇总。

每一个人的原则各不相同，需要一点一滴积累，持续改进。

不容忍问题

不容忍问题，才能真正地坚持原则。

痛苦是一个人潜在的成长机会。
你将有机会获取新的经验，放入你的原则工具箱。
你处理问题的原则将越来越丰富（强大）。

就像舞台上的主持人，已经为最尴尬的情形做好彩排，就不会再有什么意外发生。

理解和应对不好的东西才是更重要的，好的东西会自己照顾自己。

总是为最糟的情况做好准备，便能常常超越预期，越来越好。

> 自然的一项根本法则是，为了赢得力量，人必须努力突破极限，而这是痛苦的。

当真相与愿望不符时，大多数人抗拒真相，逃避痛苦。

但是，没有痛苦就没有收获。

我们的潜意识倾向于把一切不了解不愉快的东西都视为威胁。
你应该遵从理智——更高级的你，直面问题，不逃避问题。
甚至，培养 " 从犯错中学习的愉悦感 " 。

> 如果你能养成一种习惯，面对精神痛苦时能够自动地反思痛苦而不是躲避痛苦，你将能够快速地学习和进化。直面你的问题、错误和弱点导致的痛苦现实，会大幅提高你的效率。

与人合作

> 成功有两条路：（1）自己拥有成功所需的要素；（2）从其他人那里得到成功所需的要素。

当我们在一个陌生的地方迷路了，我们知道最好的选择的是向其他人求助。
但是在生活中遇到的很多问题，我们都更倾向于自己解决。

比如，我们会担心别人觉得自己无知，也可能刚刚在他人面前夸下海口……
总是因为这样那样的原因羞于启齿。

因为担心自我形象，让别人的看法，成为自己的障碍，事后来看，都是很愚蠢的。
屈服于潜意识的情绪，就相当于放弃了选择，把一切都交给更低级的自己。

当你可能有情绪的时候，先不要做判断，不要只考虑直接结果，要考虑后续与再后续的结果。
特别是，当有多个可信的人都说你错了，那就要认真考虑自己是不是真的错了。

比起目标的达成，我们是不是总是正确的，是不是总是由我自己来完成的，并没有那么重要。

每个人都有弱点，都有自我意识障碍和思维盲点。

> 大多数人犯下的最大错误是不客观看待自己以及其他人，这导致他们一次次地栽在自己或其他人的弱点上。

我们也没有足够的精力了解所有的事实。

> 与你知道的东西相比，能不能妥善处理“不知道”才是更重要的。

认识人与人的不同，才能真正从他人角度看问题。

与目标一致的人合作，才可能找到事实上最好的解决方案，而不是你自己可以想出来的最好方案。

愿我们都有勇气做出最好的抉择，成为最真实的自己。

人生中最重要的事，是洗碗

一伙人有说有笑去到火车站，却发现进不了站。 票买错了，还是昨天的，所以也无法改签。 激愤之下，行程取消。

负责买票的小妹一脸委屈： 我实在是太忙，一整天都很忙，只用了几分钟买票，没想到……

不，你会出错是因为你不够重视，你认为这件事情不重要！ 如果你足够重视，自然会放在心上。

那么，什么才是重要的？ 人生中最重要的事，到底是什么？ 突然忍不住想。

小时候，一开始是玩，后来是学习、工作…… 再后来，是生老病死。

上学的时候，努力读书， 上班之后，努力工作， 然后结婚生子，努力做一个好丈夫、好父亲、好儿子……

这些，都很好。 可是，却有些入戏太深。

比如，读书的时候，真的就只是读书吗？ 工作以后，心里就只有工作吗？ 有了小孩之后，就都成孩奴啦？ ……

想想这十年，仿若弹指之间。

还记得有位老板说：别紧张，这只是一份工作。 我当时是一脸无法置信的表情。

现在，才渐渐明白。 不是说工作不重要， 而是，工作之外，还有很多很多……

你那么努力地工作，还不是为了生活。 可是，你到底有没有好好生活过？

你说等你有钱了，你就…… 你说等你有时间了，你就……

可是，在这之前，许多个日日夜夜， 你过得像垃圾一样……

即使是生老病死，也没什么好纠结。 人生中无法控制的事，绝不是最重要的事。

人生中最重要的事，是当下我们正在做的事。 人生中最重要的事，是我们每天都在做的事。 人生中最重要的事，是柴米油盐酱醋茶。

想到这里，忽然发现我有一个极大的优点—— 当我工作的时候，我是很认真地在工作。 当我洗碗的时候，我是很认真地在洗碗。

好了，不多说了，还有一堆碗没洗。

今天，我读了一首诗

今天，我读了一首诗

然后，又读了第二首

第三首，第十首

.....

十几年了

回忆当初自己

诗人的样子

却是怎么也想不起

.....

包括那些女人

那时还是女孩

他们送的手表

和情书

.....

十几年了

我从未打开我的抽屉

或许我从来就

未曾有一个抽屉

.....

也未曾有过钥匙扣

和其他一些

杂七杂八的东西

.....

我凑几年钱买的单车

第一次从农村开到城市

为了音乐和书

我的车就被偷了

.....

我的唯一的足球

却被我一脚踢爆

我转身就走

没有回头

.....

今天，我读了一首诗

然后，又读了第二首

第三首，第十首

.....

然后，我什么都不记得了

《公司理财师专业能力》笔记

《公司理财师专业能力》—— 陈玉罡 王大中 史欣向

一、公司理财建模基础

复利

终值、期数、现值和收益率
Excel: FV NPER PV RATE
FV=PV*(1+r)^n

- 72法则：

投资翻倍所需时间，如收益率为8%，翻倍需72/8=9年

- 115法则：

投资三倍所需时间，如收益率11.5%，三倍需115/11.5=10年

所以，如果收益率达到24%，投资3年可翻倍，收益率达到23%，则5年三倍。

- 有效年利率i，年限n，计息次数m
  i=(1+r/m)^(n*m)-1

年金

Excel: PMT（Type=1表示期初，默认期末0）

年金终值 FV=PMT*((1+i)^n-1)/i
后一乘数为年金终值系数

年金现值 PV=PMT*(1-(1+i)^-n)/i
后一乘数为年金现值系数

- 预付年金
  FV=PMT*(((1+i)^(n+1)-1)/i-1)
  预付年金终值系数=(n+1)普通年金终值系数-1

PV=PMT*((1-(1+i)^-(n-1))/i+1)
预付年金现值系数=(n-1)普通年金现值系数+1

- 永续年金 PV=C/i

投资决策与融资决策

决策维度：现值、终值、收益率、年限、NPV
Excel: NPV

分期付款

- 等额本息

PMT=PV*r*(1+r)^n/((1+r)^n-1)
每期利息=（贷款总额*累计已还本金）*期间利率*
每期本金偿还额=*每月还款额*每期利息

- 等额本金

每期本金 PRN=PV/n
每期利息 INT=BAL*i% （BAL尚未偿还本金）
每期还款总额 PMT=PRN+INT

二、财务报表分析

1、盈利能力指标

- 毛利=营业收入-营业成本
- 毛利率=毛利／营业收入
- 营业利润率=营业利润／营业总收入
- 净利润率=静利润／营业总收入
- 总资产回报率=净利润／［（期末资产总额+期初资产总额）／2］
- 净资产收益率=净利润／［（期末所有者权益+期初所有者权益）／2］

2、营运能力指标

- 总资产周转率=营业收入／［（期末总资产+期初总资产）／2］
- 存货周转率=营业成本／［（期末存货+期初存货）／2］
- 应收账款周转率=营业收入／［（期末应收账款+期初应收账款）／2］
- 平均收款期=应收账款／（营业收入／360）
- 固定资产周转率=营业总收入／［（期初固定资产+期末固定资产）／2］

3、偿债能力指标

- 流动比率=流动资产／流动负债
- 速动比率=（流动资产-存货）／流动负债

- 利息保障倍数=息税前利润／利息支出
- 息税前利润=净利润+所得税+财务费用
- 现金偿付比率=（息税前利润+非付现费用）／利息支出
- 资产负债率=负载总额／资产总额
- 长期负债比率=长期负债／资产总额
- 长期负债与资本总额比率=长期负债／（长期负债+优先股权益+普通股权益）
- 债务权益比率=负载总额／权益总额
- 长期债务权益比率=长期负债／权益总额

4、成长能力指标

- 每股收益增长率=当年每股收益／上年每股收益-1
- 每股总营业收入增长率=当年总营业收入／上年总营业收入-1
- 每股营业收入增长率=当年营业收入／上年营业收入-1
- 净利润增长率=当年净利润／上年净利润-1
- 资产增长率=当年资产／上年资产-1
- 股东权益增长率=当年股东权益／上年股东权益-1

5、现金流量指标

- 回收现金能力：销售商品提供劳务收到的现金／营业收入
- 利润质量：经营活动产生的现金流量净额／净利润

- 杜邦分析

ROE=净利润／所有者权益
=净利润／所有者权益*营业总收入／总资产*总资产／所有者权益
=净利润率*资产周转率*权益乘数

ROE=总资产经营利润率+负债／所有者权益*（总资产经营利润率-财务费用率）*
=*总资产经营利润率*财务杠杆率*差异率

- 财务评分
- 雷达图

三、公司投资分析

- 折旧：sln, ddb, db, syd, vdb
- 项目评估：NPV，IRR，MIRR，方案管理器
- 债券估值：PRICE
- 债券收益率：YIELD
- 零息债券收益率：DISC，YIELDDISC

- 股权投资估值模型：

股利贴现，现金流贴现，权益现金流贴现，经济增加值（EVA）贴现，剩余收益模型，调整现值APV模型，实物期权法，相对估值法

四、公司融资分析

- 贷款融资：IRR
- 债券发行价：PV
- 发行股票：现金流贴现，市盈率模型，市净率模型
- 配股增发
- 认股权证
- 优先股
- 可转换债券
- 债券置换
- 融资租赁
- 风险投资
- 短期融资：

无抵押贷款，抵押贷款，信用融资，应收账款融资，信托贷款，短期融资券

五、营运资本分析

1、现金管理

目标现金余额：鲍莫尔模型，Miller-Irr模型

2、信用管理

- 信用政策评估（Excel：模拟分析，单变量求解）
- 授信决策
- 现金折扣政策

3、存货管理

经理订货量（EOQ）模型

六、并购分析

- 标的选择
- 公司估值：现金流贴现，市盈率，市净率，市销率

- 决策分析：现金并购，换股并购，协同效应

关于如何佛系地辅导作业

谁的作业

听说小区有位家长辅导作业后高血压入院了。 这是首次在现实中，而不只是朋友圈，得知的悲惨遭遇。

脑海中不禁浮现出这位朋友站在阳台上独自凌乱的表情…… 还有他之前说过的那句话—— 我这一世英名，都毁在你手里了！

我们小时候光环四射的童年，和现在熊孩子的完全不care，形成了鲜明的对比。

老师也说，孩子态度有问题。 考得怎么样，他完全不在乎，连考试都在玩！

如此冷静，仿佛已经堪透一切。 （我觉得，这孩子不简单。）

我家的小孩，其实也好不到哪里去。 粗心大意就算了，关键是，看上去好像是 故！意！的！

每天都听到大嘴妈在大吼大叫，硝烟滚滚。 换我去，过了不到一个星期，老师投诉。 作业质量不过关，不听课…… 于是连我也成了批斗对象。

但我依旧佛系。 我的口头禅是： 又不是我的作业，我紧张什么？

因果

事实上，我早就觉得老妈有些越俎代庖了。

饭盒没带，回来劈头就冲老妈喊： 你怎么不帮我带！

东西不见了。 你怎么不帮我找！

作业漏做了。 你怎么不告诉我！ ……

反正都是老妈的事。

当然，他不敢冲我喊。 （我一棍子抽过去。）

我以为，人类最朴素的经验，是从因果中学习。

他虽然承受了一定的结果，但是当他追溯原因的时候，结论却是： 老妈（别人而不是自己）没做好。 这样他永远也不知道问题出在哪。

巨婴的长成，不外乎此。 （我也不想，因为XXX，所以我没办法。）

界限

然而，有些事情，只有真正成为了父母，才知道其中的艰难。

父母对子女的爱是没有界限的。 然而我们又要试图告诉孩子，处处有界限，有规则。

你再这样，下次看我不打断你的腿！ 但你根本不可能打断他的腿。

所以下次当你再这样说的时候， 就再也没有作用了。

对孩子也是一种煎熬， 因为有时会挨打有时又不了了之， 只能小心翼翼察言观色。

孩子会看你怎么做而不是怎么说。

所以，父母首先要知行合一。

我觉得，这一点我还是做得比老妈好的。 当我说打的时候，我是 真的打。

个体

虽然电影里都说，你不养我，为什么要生我？ 但反过来，包办一切，也不会有好结果。

孩子虽然是你生的，但是他却不是你的私人财产。

孩子是一个有独立思想的个体。 他们是和你完全不一样的人。

孩子很可能不会继承你的优点，遗传具有随机性。 而且，他们大概率会不如你，假如你很出众的话。 （当然，假如你比较笨，孩子大概率会比你聪明。） 何况，你的所谓的优点，大都来自你后天的努力和际遇。

我们终身奋斗的目标，在孩子看来，也很可能一文不值，孩子一定会有他们自己的目标。

特别是，还有死亡，生命最伟大的发明，它确保老的总是让位于新的。

我一点都不担心，孩子会比我们差。 我唯一关心的是，该怎麽告诉他们： 为什么今天的生活来之不易， 如何坚强面对明天的挫折， 怎样满怀信心和勇气开创未来。

这一届小孩，幸福得跟花儿一样。

妹妹画画：喜欢即是欢喜

小美女又要去画画，

拿了一把小伞，在我面前晃了一下，就消失了。

.....

等我从房间出来，已经不见人影。

赶紧穿鞋子下楼。

.....

等到楼下，却是一个人也没有。

天，阴阴沉沉的，潮湿的地面泛着光。

.....

电话回家再问哥哥，说确实已经走了。

惊出一身汗，赶紧追出去。

.....

终于，在小区门口看到她，

一个人，撑着一把蓝色小伞，静静地等着。

.....

听说今天要做灯笼，所以小朋友来得特别多，

熙熙攘攘的，还有些大喊大叫。

.....

这些小孩，真正喜欢画画的不多，

有些画不到一会儿，就开始张望和走动。

.....

还有一个，时间到了还没画完，

哭着要妈妈帮他画，没完成又不肯走。

.....

我家小美女，却是真的喜欢。

她就坐在那里独自静静地画，从不管旁人。

.....

画，即是欢喜。

想不到古人已经在用护手霜了，还是专业护手霜

今天重读庄子《逍遥游》，就是那篇："北冥有鱼，其名为鲲。鲲之大，不知其几千里也。化而为鸟，其名为鹏。鹏之背，不知其几千里也；怒而飞，其翼若垂天之云……"

里面有一个护手霜的故事，比较少听说，记录一下，以飨大众。

古人怎么会有护手霜呢？还真的有，原文是：宋人有善为不龟手之药者。

龟，即龟（皲）裂的意思。不龟手之药，确定无疑是护手霜了。

这个有护手霜秘方的宋人家族，世世代代以漂洗棉布为生。有一天，有一位客人听说了这件事，愿意出百金购买他们的秘方。于是族人集合起来商量道：我们几代人洗布，累积所得不过数金，现在立刻就能得到百金，不如还是卖了吧。客人拿到秘方，用来游说吴王。那时刚好越对吴发动战争，吴王就命他带兵抵抗，在冬天与越人水战，大败越人，吴王最终封给他大片土地。

同样是护手霜，用途却大不相同，有人用来洗布谋生，有人凭此成封疆大吏。

我们应该知道我们所应该知道的

关于新冠肺炎的一些思考，仅代表个人当时当刻观点，供记录回顾之用。

这几天人们对于流感疫情的反应有些过了，谣言满天飞。
从死亡的病例来看，基本上是带有多种并发症的老人家。
SARS的时候，死的许多都还是年轻人，触目惊心。
所以根据我当下及其不科学的判断，这还只是一个稍微厉害一些的流感。
既然可以通过增强体质熬过去，年轻人其实不应该那么恐慌。

看到网上发的门诊排长龙的照片，有些可悲。
这其中到底有多少是普通感冒发烧的，不做一个基本的判断就去排队，给自己增加了多少感染的风险？
充满病人的医院当下是最大的传染源，这是第一个重要的基本事实。

这不是危言耸听，根据科学统计，当自杀事件被大肆报道的时候，自杀率都大幅上升，何况流感这种影响范围极大的公共事件。
你当然可以责怪医院为什么不派人出来先做一轮快速筛查，但身体是自己的，我们应该根据现有信息做出明智的决定，真正对自己负责。

当人心惶惶，情势未明时，最可靠的人是你自己。

当然，病毒本身还可能变异，所以阻隔传播还是十分必要的。
这次我所看到的广州，反应是及其迅速的。
第一天新闻出来的时候，三号线已经有两三成人带口罩，第二天就到了五成，那时我的判断还是乐观，因为看到工作人员还是没带，第三天就接近百分之百了，接着就是省一级响应，开始量体温。
现在高铁上，也没有看到人随意把口罩扯下来。

文明都是培训出来的。
以前我们看到人随地吐痰，虽然无奈，但也知道这样的人还很多。
但是现在假如在人多的地方还这样做，相信大家都会看傻X一样看着你。
大家带口罩的时候你没有，也显得很不礼貌。

这不是素质而是习惯的问题，简单来说是因为社会进步得太快了。
中国占大多数的还是农民，城市里相当一部分人在十几二十年前还是地地道道的农民。
你让一个农民吐痰到纸巾里包起来再找垃圾桶去扔，想想就不是容易的事情。

闯红灯，也是相同的道理。
我是这样劝老父亲的——按照现在的法律，违反交通规则被撞，是不会赔钱的，一分都不会有。
老父亲看重钱物，饥荒年代的童年，动乱时代的青年，我知道钱对他意味着什么。

基于事实和常识，我们应该知道我们所应该知道的。

2020-01-25 年初一于广州南动车上

截至2月3日 15:20，确诊17252，死亡361，死亡率2%。
湖北确诊11177，死亡350，死亡率3%，即除湖北外其他地区死亡率0.18%。

以上数据似乎印证了我之前的推测，这只是一个传染性稍微厉害一些的流感。
除湖北外，其他地方的死亡率可能低于普通流感。

假如没有恐慌和混乱，像普通感冒一样，我们感觉不太好了再去医院，这样每个重症病人都能得到充分的照顾，更避免了严重的交叉感染，相比当下局部的失控，结局可能会好很多。

大多数的天灾，说到底都是人祸。

每当遭遇公共危机，个人下意识的行为都是逃离现场。
事后看来，这似乎是一个相当明智的选择。

但是，没有假如。

2020-02-03 年初十于电白

我在清河市场买菜

人们常说，当你觉得苦的时候，去医院看看，去殡仪馆看看。
但是，我觉得，还有一个地方，很值得去，那就是——菜市场。

此刻，我正在清河市场，看人来车往。

这里，绝对是全广州最大的菜市场之一，特别是每逢四九墟，夸张到连手机都没信号。

这里，充满了烟火味，饮食男女，熙熙攘攘，忙忙碌碌，有老有少，有贵有便宜。

突然想起以前网上看的，周润发坐巴士去买菜的情景，新闻上说他身家几十亿。

又想起前两天李克强总理说的，中国还有6亿人，月收入不到一千。

感觉网上新闻还是少看的好，消息越灵通越感觉魔幻。

直播间的小哥哥小姐姐，也非普通人模样。

而且看到人们都在为欧美发达国家担心，美国政府更是整天指指点点，生怕我们哪里发展得不够好。

不过都没什么大不了的，人生无非就是吃吃喝喝。

平凡人，有工作，有饭吃，便是太平盛世。

没有什么是一顿饭解决不了的，不行，那就两顿！

以前，一直觉得，人生挺荒诞的。

后来，才发觉，一切奇怪的现象背后，无非都是吃饭问题。

饭吃好了，什么都解决了。

当然，有些时候，吃着自己的饭，却总觉得别人的比较香，还怀疑别人的食材比较上乘。

实际上，五毛钱一个的鸡蛋，跟五块钱的相比，并没有多大不同。

关键还是看手艺。

你只会水煮，别人却可以做荷包蛋、番茄炒蛋、蛋挞、蛋糕……

同样几块钱，不同的味道。

漫步华尔街

> 唯一比死更糟糕的事，就是人活着，钱没了。
>
> ——《漫步华尔街（原书第11版）》

能够十几版的书，想必不会太差，果不其然。
三观很正，或者说，跟我的想法很接近。
而且这本书讲的主要是历史，也就为我们的观点提供了很多支持。
我们从来不缺乏观点，缺的只是证据。

先存钱

在你有一定的积蓄之前，谈论10%，15%，是没有意义的。所以年轻人首先要努力工作，积攒本金。
你的风险承受能力，主要取决于你在投资之外的现金流。一般来说，主要来源于你通过劳动可以换取的收入。
存钱，是积累本金，投资自己，是提高积累本金的速度。

所以，年轻人最好的投资，是投资你自己。

说来惭愧，炒股炒了十几年，才明白这个浅显的道理，幸亏没有投入太多精力，不然，人就废了。

随机漫步

特别是不要幻想可以找到股价涨跌的奥秘。
股价的短期走势无法预测，是一种随机漫步。
大量研究数据表明，所有我们熟知的策略，都无法获得超额收益，包括技术分析、基本面分析、大市值、小市值、聪明β、低波……
它们都只能在一段时期内跑赢全市场指数。

这里其实隐含了一个意思，市场最终是有效的。

但市场也不总是有效，因而套利机会还是会存在。
捡钱的机会不多，所以最好做好准备，一旦遇上，赶紧把它捡起来。

长期向上

既然股价无法预测，那为什么还要投资呢？
因为股票资产作为一个整体，它的长期趋势是向上的，大大跑赢债券和房产。

书里推荐的组合，股票基本占五成以上，而且风险承受能力强的，应该适当增加小市值股票占比。

不过，这里的长期，指的是十年甚至二十年以上。

> 很多投资产品中所蕴含的风险，会随着投资产品持有期的延长而逐渐降低。

在震荡或跌多涨少的市场里，长期定投可以有效降低股票和债券的投资风险，提高收益。

指数投资

当然，这里所说的投资，主要是指数投资。
每一个投资组合的核心部分都应该实现指数化，即主要投资指数基金。
比如推荐的35~45岁的投资组合，配置如下：65%股票基金，20%债券基金，10%房产信托及5%现金。

> 到目前为止，尚无任何科学证据表明由专业人士管理的投资组合作为一个整体，其表现较涵盖范围广泛的指数更为出色。

所以，选择投资主动型管理基金，似乎也没有必要。

然而，大A的情况还是有些特殊，上证指数就貌似有bug，一直原地踏步。
而且，全部投资指数基金，也放弃了当前来说还相当可观的打新收益。
我的想法是，先构建一个最低的打新组合，再长期定投指数。
当然，这样一开始风险会高得多，也考验选股能力。

选股规则

1. 连续五年以上盈利增长和诚信经营记录。
2. 低估买入，市盈率与前景相比较低，与市场相比合理。
3. 买入有故事（想象力）的股票，容易在其增长预期之上建立空中楼阁。
4. 长期持股，尽可能少交易。

除了盈利增长，诚信经营同样（或更加）重要，特别是在造假成本比较低廉的市场。

另外，尽管低估可能会继续低估，但是，相比高估值股票可能要承担估值与增长的双重下跌，风险还是少得多。

股债平衡

既然股价短期波动无法预测，那什么时候卖？

> 撇开股市长期向上的趋势不谈，连续出现股票收益过高的情形并不会持久，这之后的未来收益一般而言会走低。均值回归总是存在的。
>
> 我们都希望有个小精灵能可靠地告诉我们如何“低买高卖”，而对资产权重进行系统性调整的策略，不啻是我们所拥有的那个可信的精灵。

股债平衡之所以是一个好策略，是因为它始终以风险为锚，只需关注股债比例，而无需跟踪股价高低。

至于调整股票组合时该卖哪个？ 除了考虑估值和前景，一般而言，卖掉赔钱的，留下赚钱的。所谓截断亏损，让利润奔跑。

而各项资产的目标权重，取决于个人的风险承受能力。最好按生命周期投资，年轻时多投资股票，退休后增加债券和房产。

一个简单的判断标准是，组合的波动是否影响了你的睡眠。

> 每个投资者都必须在自己吃得好与睡得香之间找到自己愿意接受的平衡点。

财富自由

本书提供了一种“4%解决方案”，即你每年花费的钱，不应该超过你已积攒下来的资产总值的4%。
这样，你的盈利才能满足支出，同时确保剩下的资产不会贬值。
这里的假设是，预期收益率为5.5，通胀率1.5。
此规则似乎对我们也同样适用。虽然我们的通胀率可能到3，但是期望收益率极可能大于7。

所以要实现所谓财富自由，只需要：

1. 尽早开始储蓄和定投，提高总资产。
2. 低欲望极简生活，减少花费。
3. 提高投资能力，收益率高于市场平均水平。

对于大多数人来说，减少花费和适当储蓄，已经可以实现体面的退休生活。
期望超越市场，反而是最不现实的，这是一个概率问题。
但人性贪婪，炒股游戏中，大都心存幻想，往往投资变赌博，最终一事无成，还不如一开始就不参与。

股市有风险，投资需谨慎。

爷爷血压飙升入院，看完之后却更不舒服了

爷爷血压飙升入院，

各种排队交费测体温量血压，

一番折腾，终于找到床位躺下，

已经几个小时过去了。

.....

护士姐姐过来准备输液，

一量，血压已经降下来。

.....

爷爷说一直担心接不到孙女，太紧张了。

头晕之后，血压一时又降不下来，担惊受怕。

现在放松了，也就没事了。

.....

也不能一直这样干等着。

只好收拾东西回家。

.....

一路上我都在吹牛，说：

根本就不用掏钱，我医保交了多年，

又从来没有用过，卡随便刷。

.....

不过回到家，血压还是升了。

估计一直还想着临走时医生说的话：

药是不用了，可是钱，也是不能退的。

.....

唉……

生又何哀，死又何乐？庄子曰：至乐

庄子前往楚国，看到一个骷髅，还是很完好的头颅形状，于是用马鞭在旁边敲了一下，凑过去说：你是因为贪图享乐违反天理，而至于此吗？还是有亡国之事，死于战乱武器？或者是你做错了什么？觉得羞耻，愧对父母妻儿，所以自杀？还是因为吃不饱穿不暖，贫病交加而死？又或者只是活得太老，寿命到了尽头？

庄子说完，把骷髅带了回去，睡觉时把它当成了枕头。夜半，骷髅出现在梦里，说：听你说话的语气，像是能言善辩之士，不过我看你所说的，都只是活人的拖累，死了，就没有这种烦恼了。你想听听死人的说法吗？

庄子说：好的。

骷髅说：死了以后，无君于上，无臣于下，也不分春秋冬夏，纵然自如，以天地共长久，即使南面为王，也没有这么快乐。

庄子不信，说：我请司命复活你的形体，恢复你的骨肉肌肤，返还你的父母妻儿、亲戚朋友，这样你想要吗？

骷髅皱眉蹙鼻，一脸愁苦地说：我怎么可能放弃南面为王的至乐而重新经历人间的劳苦呢！

生命是最基本的人权

新冠病毒确诊已经超过一百万了，却依然有如此多的所谓民主国家在忙着质疑中国数据造假，（之前嘲笑的是方舱医院、隔离制度），在争论口罩有没有用，或其他东西，实在是人类悲哀。

转移注意力，对于当下抗击病毒挽救生命，并没有什么实质的作用。
武汉的悲剧一再重演，说明人真的很难从别人的经验里学到什么，自己经历过，才真正有切肤之痛。

而一个国家或地区稳定的关键，不但取决于制度的形式，更重要的是如何保证上位者由全社会的精英组成，能够迅速达成共识，即如孙中山先生所说的专业的人做专业的事，这才是最好的政治。

而选择精英的标准，不是他（她）有多成功，而是有多智慧，因为无论多完善的制度，（比如上市公司董事会），都无法防止聪明的人造假。

一个人最重要的品质是诚实，诚实面对自己，诚实面对所有问题，能迅速纠错。

而人类作为一个整体，善良是最基本的底线。

那一个个统计数字背后，是千千万万个破碎的家庭，不分国家，无论种族。

生命是最基本的人权。

想起读《宽容》时的笔记，再复习一下。

宽容

1. 宽容的定义

允许别人有行动与判断的自由，对有别于自己或普遍的见地要耐心公正的包容。

2. 不宽容的根源

2.1 生存基因

恶劣的环境使得人类必须给自己制定严厉的律法，以保证整个族群顺利地生存下来，其根源是对自然力量的敬畏和对无知的恐惧。

对规则的破坏也一定曾给人类带来过几乎毁灭性的惩罚，这种惨重的代价刻骨铭心，深入基因，使人再不敢越雷池半步。

2.2 人的社会性

人是群居动物，一个人只有当感觉自己属于某个组织或群体，而且群体里的人也都秉承相同的偏好、恐惧、理想和信仰时，这个人才真正感到幸福。

正是这些趋同性使人不顾艰难困苦和危险聚在一起，共同排斥那些少数与他们不同的人。

3. 如何应对不宽容

对不愉快的事情生气无任何价值，因为事实完全不在乎我们的感受，所以也不会改变。

我们所能做的是，保持冷静的头脑，随时做好应对（战斗）的准备。

老婆死了，庄子却还在唱歌，这到底怎么回事

庄子老婆去世了，惠子前往吊唁，发现庄子正在敲着盆唱歌。

惠子说：你妻子与你结婚，一起生活多年，养育子女，慢慢变老，现在去世了，你现在不哭也就算了，还鼓盆而歌，这不是太过分了吗！

庄子说：事情不是这样的。她刚刚离开的时候，我怎么能不感慨悲痛！可是后来仔细想想，认真追究起来，她原本就是不存在于这个世界上的，不仅仅没有生命也没有形体，不仅仅没有形体也没有气，处于天地混沌恍惚之间，然后变而有气，气变而有形，形变而有生命，直到现在又回到死亡，这就跟春夏秋冬四季变换运行一样自然。现在她人已经安然躺于天地之间，而我却在这里嗷嗷大哭呼天抢地，这又成了什么样子？想来不合于天命自然，所以我就想开了。

人的生死，本来只是一种自然现象，奈何活着的人却总是看不清，堪不破。

读《未来简史》，思考未来

《未来简史》这本书，值得推荐，不是说写得多好，而是说，我们太缺乏这种思考未来的书了。回头再看看《人类简史》，赫拉利成名之作，应该更好。

想一想，我们之中有谁，思考过我们人类的未来？或许甚至连我们自己的未来，也不敢想太远。就像投资一家公司，我们一般也只能看三到五年，三五年之后，谁知道呢？

不过这些年来，我常常给老人（自己）洗脑：我们理解不了这个世界，是因为这些年中国的变化太大了。变化太快，所以很多原本很有价值的东西，也变得没那么重要，比如家里屯的粮食，一份旱涝保收的稳定工作……对应的现实是，以前吹牛说亩产一万斤，实际上只有两三百，现在却确确实实有一千多甚至两千以上，而且需要的人工更少。普通人肯付出，很容易就可以找到工作，解决温饱基本上没有多大困难。

认识到社会的变迁，一个人才不会太执着于已有的东西——我称之为迷信。我一直在试图破除迷信。没有什么是一成不变的，放下过去，才能享受当下，看见未来。

《未来简史》把近现代史归为人文主义的历史，结构上还是很清晰的。无论个人自由主义，进化自由主义，社会主义，本质上都是说以人为本，人定胜天，达到所谓人化身为神。后面提出数据主义，把不同的社会结构和宗教（人文）归结为数据的收集、分析和处理的方式和效率问题，也有一定的启发意义。

突然想到，西方所理解的社会主义，和我们的理解是不一样的。马克思主义，除了说提高无产阶级的福利，在当时的历史条件下，更多强调的是阶级和暴力，所以社会主义很容易被标记为专制和集权。但是我们现在所说的中国特色社会主义，除了满足最广大人民群众对美好生活的追求，更加强调的是人民至上。用毛主席的话来说就是：实事求是，为人民服务。毛泽东思想比马克思主义的内涵要丰富得多，包含了传统的智慧。

包括中国饱受诟病的选举制度，也主要是来自于历史的传承，今天看来，比西方民粹主义也要明智得多。一人一票，结果几乎肯定是赢者通吃。科举加选举的制度，使得大量的公务员都来自于基层，反而更加公平和有代表性。孙中山先生曾经说过，政治应该由专业的人来做，现在似乎已经是这样。先生《建国方略》里的规划，今天也已基本实现。

有公知说中国人没有信仰，这是很可笑的。信仰不一定是指传统的宗教。卢梭有句名言：人生而自由，却又无往不在枷锁之中。每个人都生活在社会和自己编织的网里，每个人都有自己的宗教。赫拉利认为，现在最大的宗教，其实是人文主义。

中国人信仰的是中华文明，具体来说，是道德。这个道德，指的是《道德经》的道和德。中国人，无论嘴上说什么，当我们要做什么决定的时候，首先考虑的一定是道和德：是否符合天道，是否符合仁义。我们常说谋事在人，成事在天，这里所说的天，并不是指上帝，而是指是道法自然的自然，即客观规律。我们平时拜的神，也不是上帝，而是祖先，是文化传承。

这样回过头再看毛泽东思想，就会发现，说的其实也是一回事。中华文明为什么能生生不息？几千年前，我们就已经奠定了强大的指导思想，然后历史的叙事便一直保持惊人的一致，一脉相承。中国的第一本经典《易》，就已经告诉我们，世界变动不居，阴阳转化，和而不同。这种认识早就已经深入骨髓，活下去自然不成问题。中国人很早就已经开始追求道法自然，天人合一，换成现在的说法就是，科学和可持续发展。

不过我估计，中国真正的文艺复兴，应该要到零零后以后。八零后九零后，虽然没有一些前辈那么自卑舔跪，但是我们童年成长的环境大都还是相当艰苦。唯有零零后以后，当他们站出来，一定会有一种油然而生的天然的自信。到那时，他们眼里的中国历史与未来叙事，一定会比我们靓丽得多。

个体来说，我们无力改变当前的叙事，但是我们可以改变我们的看法。关键还是要破除迷信。房子，车子，银子，面子……真的那么重要吗？未来真正重要的是什么？

不管世事如何变幻，不管现在还是未来，知行合一，就是幸福。

读《邓小平时代》

中美贸易战和科技脱钩之际，回看中美建交的历史，感慨万千。破坏关系容易，要建立起来，却非常难，无法想象的曲折和来之不易。

邓小平确实是毛泽东之后最伟大的战略家，建交欧洲日本，联美抗苏，敲打越南，继而裁军，坚持改革开放和全力发展经济，提高人民生活水平。

每一步，都是出于战略的考虑，为中国赢得了几十年的和平发展空间。

如今，国际形势急转直下，波诡云谲，再次考验各国人民及其领导人的智慧和定力。

但看完这本《邓小平时代》，却不禁使人对中国的未来充满信心。

巧的是本书的作者傅高义，正是一位著名的——美国的中国通。看了本书前言超长的考证过程和采访名单，就知道，这绝对会是一本好书，而且，美帝的实力，绝对不容任何低估。

本书作者力图客观，有时甚至是以故意淡化个人影响的笔触，使我们看到，中国今天的稳定和繁荣，虽然有赖于某些领导人，但主要还是集体智慧的结晶。

比如我们今天所看到的对文革的评价和对毛泽东思想的总结，并不只是来自少数几个人的观点，而是经过了整个高层超过几千人的广泛讨论和一步步完善。

华国锋虽然自称是毛主席的追随者，可是却第一时间就收拾了四人帮，也不搞政治运动和阶级斗争，而以务实的方式推动四个现代化。

胡耀邦和赵紫阳虽然对学生运动处置不力，但他们都兢兢业业，坚持了发展经济和进行改革的正确方向。社会主义初级阶段的提法就始于赵。

更有叶剑英老元帅携一众老将军坐镇，确保高层更迭而不失控，时局虽乱而不崩溃，居功至伟。邓小平时代的军委，书记，主席三足鼎立的格局，正出自叶帅之手，后来集体领导制的确立，也可以说是这种智慧的延续。

著名的包产到户，陈云早在文革之前就已经提出来了。他坚持保守的目标，对经济进行的几次阶段调整，有效控制了国家债务和通胀。优先发展农业和轻工业的决策，事后来看，无疑也是十分明智的。

万里在安徽进行包产到户试验的时候，有农业副部长批评，他说："看你长得肥头大耳，农民却饿得皮包骨，你怎么能不让这些农民想办法吃饱饭呢？"

改革开放在广东试验的时候，任仲夷“二进宫”，在北京受到严厉批评，他回来说：“不要搞运动，也不要指责哪个人。要坚决反对个人牟利，但也要坚定地支持改革开放。我作为省委第一书记要承担责任，我的下级就不必了。”

一国两制的基本思想，则是由周恩来提出来的。

邓小平被称为改革开放的总设计师，他使我们笃定了战略方向的选择，而所有具体的办法和思维，都是来自于人民群众的集体智慧。

当最高领导人只关心战略时，其实就是一种无为而治。

中国繁荣富强的终极依靠，是我们几千年的文化传承。马克思主义，只是很大程度上契合了我们传统辩证思维的突破口，毛泽东思想，才是真正的精髓，可以溯源到经史子集的远古智慧。

而邓小平，又是毛泽东思想最严格的执行者，不仅实事求是为人民服务，还要坚持不断用实践（甚至试验）来检验这个过程。

前人栽树，后人乘凉。回顾历史，中国共产党作为一个整体，确实代表了当时整个国家的有识之士。而精英治国，或许也是当下最优之选。

然而这种趋势带来的问题是，那些有钱的人大代表，如何代表我们这些没钱的人？ 所以从战略上来说，文化大革命的初衷，或许并没有我们当初所以为的那么错。

让精英阶层下乡放放牛，好处不言而喻。然而反过来，拉目不识丁的农民和工人来当领导，无疑是错的。人性不可挑战。缺少知识和能力的人作为上位者，所能想到用来保护自己位置的做法，只有打击报复。

但毛主席还是固执地说，他主要的功绩，一是建国，一是文革。这里面，显然也是战略的考虑。

江泽民同志的“三个代表”，似乎回答了这个问题。一党专政的国家，要保持合法性，这个党必须要代表国家先进生产力和先进文化的发展方向，更要代表最广大人民群众的根本利益。

2020年，有新冠病毒的挑战，有中美对抗的转折，但是我们的主要任务还是扶贫攻坚，国家的长期战略得到了很好的延续，初心不改。

如何面对各种挑战？且听小平同志赠言：“冷静观察，稳住阵脚，沉着应付，有所作为。”

有所作为，主要是指，做好自己。