The Initons Catalytic Reflection Between Humanoid DNA and Nero Cell

类人 *DNA* 与 神经元基于催化算子映射编码方式

Yaoguang Luo, Rongwu Luo
Liuyang Deta Software Development Limited Company
Changsha Lehao Fu Chan Hospital
罗瑶光，罗荣武
浏阳德塔软件开发有限公司
长沙乐好妇产医院

Keywords
VPCS, AOPM, IDUC, Nero, Artificial, Decoder, Medical, Paralling, Computing, Humanoid, ETL, Parser, Data Mining
关键词
***VPCS* 架构 *AOPM* 逻辑 *IDUC* 编码 神经元 人工 解码 医学 并行计算 类人仿生 *ETL* 数据挖掘 爬取**

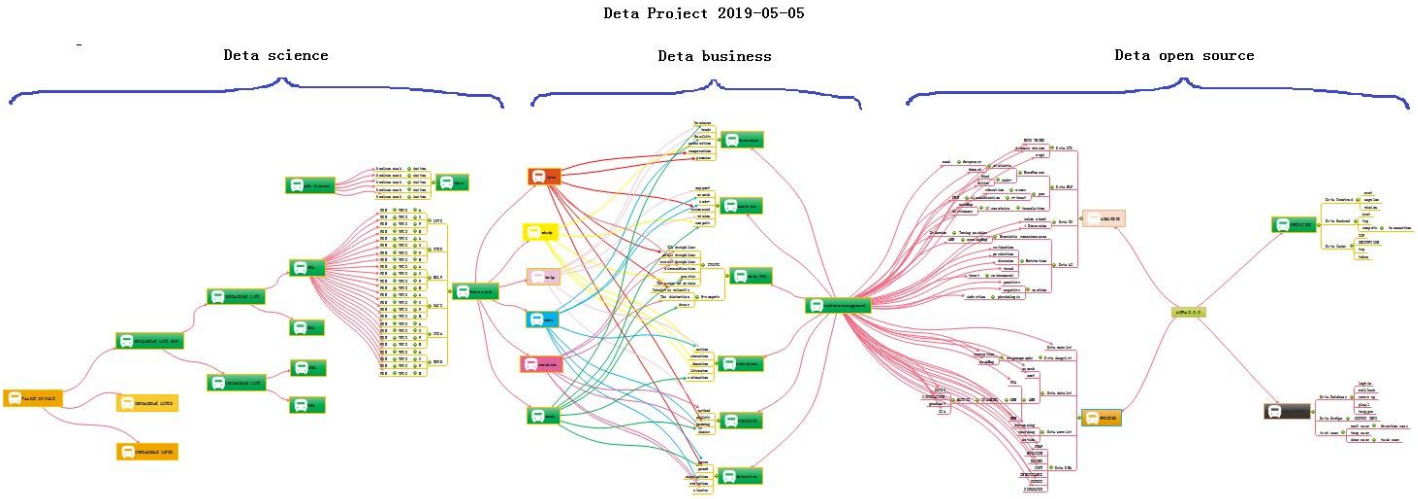
Outlook: *VPCS archetecture is not the end. Absolutly, At least at this paper, I will make an implimentation in five aspections: DETA humanoid cognition, DETA Medical Business backend logic, DETA Catalytic computing, DETA Finding initions, DETA DNA decoding. Above all I also will spend more and more words in my Deta DNA Law of IDUC. And its applications in the real world. For more informations and sources please find wechat ID: sweet00048, ok next step as below.*

观点*: DNA* 的本质是一种智慧体 对 数据 增删改查 的 四个 元操作 的 组合方式 编码*. VPCS* 架构不是德塔最终架构，观点很明确，至少这篇文章中，作者会做一个详细的论点论证,基于 *5* 个部分，德塔工作在类人认知方式，医学商业逻辑的后端处理，催化计算的过程，德塔计算算力本质的寻找，以及仿生 *DNA* 的解码分析，最后，作者会组织很多的文字来描述德塔的一些振奋人心的发现*:* 比如 *DNA* 的编码元基 如 *IDUC* 对应 增删改查，以及它的真实环境的社会应用实例.

I DETA humanoid cognition 德塔类人认知

德塔开源的起源，建立在一堆疑问之上,作者有必要进行解释，人造人？永生？自我进化？答案有很多，无疑都是正确的答案，德塔一直努力在创造一个基础数据计算体系，让技术更好的适应生活环境.通过多年的努力，德塔找到了一些振奋人心的答案，比如 对环境的认知能力，这种能力用计算机语言来描述就是 基础算法，人工智能的启蒙. 作者第一次 接触认知这个词汇,是在英特尔 福森总厂 工作的时候 接触了 *Sonar Lint* 的 认知测试，当时感觉 认知是一个趋势. 虽然 *sonar* 的作用是让代码更简洁有效，保证代码书写的规范性. 作者感觉认知方式 的 基础灵感 很多应该来自于源码的规范.

通过 *sonar* 给作者的灵感，作者一直在思考，如果 *sonar* 中的规范成分能塑造人的行为规范，那人 劳动方式和 代码的执行方式是一样能够准确定义的. 于是一个计划在作者的心里萌芽. 编写人类劳动的认知规范. 怎么写？作者有一个明确的思路，用自身对世界的认知来重塑一个规范.逐步优化它.于是 源计划 开始.



如图的的工作非常的明确，先设计开源基础,然后进行商业论证,最后催化编码设计类人进化智能.
截至到公元 *2020* 年 *10* 月 已经实现了 *18* 个德塔数据智能子工程 开源，*6* 个独立的知识产权软著， *2* 个医学大数据辅助学习诊疗系统，*AOPM VPCS* 的 *initons* 催化编码规范 *7* 篇论文. 进度有条不紊. 感谢一切. ***很多时候我所作的一切 我希望是别人所期盼的一切. 正如永生算法.***

I I DETA humanoid cognition history 德塔类人认知历史

过去，世界的认知方式 哲学家和科学家 喜欢用五感*(*触觉，味觉，听觉，嗅觉，视觉*)*来描述物体的方式，物种感觉能让生物很好的适应环境，理解环境，能够在危险的环境中思考保护自身的方法. 这些方法的实现方式丰富多彩，执行逻辑也各种各样，可是起因和结果都明确了一个基本点，**更好的适应环境**. 作者在设计养疗经的过程中，很好的融入了 语音，文字，联想，视觉媒体的仿生技术. 一直在优化它们逐渐形成一个综合智能养疗体系.

I II DETA humanoid cognition development 德塔类人认知研发

为了更好的适应环境,人类开始创造文字，发明工具，提高对事物的认知能力，从奴隶的愚昧制度到 现在开放兼容的世界，从铁器时代 到现在的纳米芯片技术. 人类 在 *5000* 年的历史中似乎是漫无目的的思维进化，透过这些现象，本质其实很容易被挖掘到，更好的适应环境和改造环境. 最有效的提高对环境事物的认知能力，最好的论据是 基础科技的研发和系统性归纳，马可尼的无线电报，祖冲之的圆周率，达尔文的物种起源，汉谟拉比法典，历史上出现无数卓学的科学家，思想家，发明家，哲学家，这些人似乎是夜空中璀璨的明星，逐渐，智能生物开始有足够的能力仰望太空. 探索宇宙的奥秘. 而这个足够的能力便是提高对事物认知的基础能力，至关重要. 德塔公司 的华瑞集系统 借鉴人类本能的系统性归纳方法，将医学教材作为认知基础来对医学疾病辩证，是符合科学发展的体现.

I III DETA humanoid cognition application 德塔类人认知应用

类人认知在社会科学上的应用很多，这里有很多优秀的论据，比如辅助听觉系统，大数据推理系统，天气预报预测系统，刑侦数据库系统，太多了，这无疑论证了认知模式大大提高了人类对环境的适应能力. 更确切的语言表达是 从提高适应环境能力,到逐渐进行局部到整体改造环境的能力. 这里又有很多卓越的论据，从上古时代的大禹治水 到 隋炀帝运河改造 到 习近平总书记的 '绿水青山' 改造，从人工降雨 到 人工岛屿，论点非常明确，对事物的认知能力来自基础科技的积累. 这些基础科技逐渐形成一个系统，在它之上是广泛的科技商品应用，如改善物资环境的袁隆平团队的杂交水稻. 改善健康环境的钟

南山团队的新冠病毒综合治疗方案. 改善地理环境的长江三峡巨型水电站. 等等, 太多了. 人类的智慧进化逐渐形成一条明确的路线, 提高基础科技与认知能力相辅相成. 这些能力 都来自对事物的问题 产生 思考, 然后形成 解决方案, 最后落实, 这个过程作者归纳为分析, 操作, 处理, 管理的 过程. 软件工程的生命周期在这里有很好的解释. 分析 *A*, 操作 *O*, 处理 *P*, 管理 *M*, 用简单的话语来描述即使将未知数据进行采集分析, 然后进行事物化操作, 操作过程中遇到的各种困难的解决方案落实, 最后维护和管理这些落实经验. 德塔的后端计算模式和系统的生命周期从最早的 采集 分析 操作 整理 编码 运行 调试 维护 逐渐浓缩 成 分析 *A*, 操作 *O*, 处理 *P*, 管理 *M* 的 模块模式, 比如 德塔分词, 德塔 *DNN* 读心术 等, 现在德塔的 *ETL* 也准备往这个方向走. 作者去年设计了一篇论文很好的描述了 *AOPM* 的应用机理如下:

AOPM Open Source System On SDLC Theory

Mr. Yaoguang. Luo

Liu Yang Deta Software Development Limited Company, Hunan, China,

313699483@qq.com

Outline: *Mr. Xuesen. Qian once said: Science Is A Titan System, as an open source software conception. this topic implements a software interaction theory of SDLC for Analysis, Operation, Process and Management— AOPM. Also, this is a tiny paper where easy to show more idyllic landscapes of using Deta open source projects. Not only for web system, also for mobile and desktop platform. The final goal are makes complex project to simple. Ok let go and the next steps.*

观: 钱学森先生曾经说过 科学是一个巨系统. 结合开源软件的设计理念, 这篇论文提炼出 软件生命周期的核心部分,如采集分析,细化操作,执行处理和运维管理的四个部分, AOPM, 这篇小论文不仅适用于开源工程, 在互联网或者桌面系统上,甚至适用于各种复杂的巨系统.

Keywords: ***SDLC, AOPM, VPCS, WEB, Concurrent, Open Source, Interaction, Management, Automation***

关键词:软件生命周期, *AOPM, VPCS*, 互联网, 并发, 开源, 交互, 管理, 自动化

Introductions

Recently my colleagues take more care on the SDLC evolution of open source software engineering, for each project they undertake on where it cost a lot of times, that's for my job, continuing found out a high effect, simple and clear theory of SDLC what be my main task now. after imagination and logic recursion, the key is an optimization of ordinary SDLC such as water fall. First time for makes an introduction to waterfall of SDLC? The author's explanation likes sequence linked list of component nodes. With Deta projects here contains four aspects at Figure1-1. And my explanation of open source as belows

介绍: 最近,我的朋友总是关心软件生命周期的演化, 这些琐碎的工程事物,不仅浪费时间, 也降低效率. 持续的寻找一种高效的清晰的生命周期,纳入了我的个人日常工作范围. 通过大量的细化和 传统的基础理念优化, 我觉得软件生命周期的瀑布模式是一个很好的参照点, 可以进行发掘归纳如下.

Topic: Ten Definition of The Open Source, OSS Book Reading Note

这是我在路德大学研 *3* 写的读后感作业

In this paper, through a premise: the contrast between the copyright and the contract. the author talks a comprehensive introduction to the definition of the open source code. The role of the open source licenses, which is to allow the work permit under the non-exclusive business. Not only does it mean that the source code was visited by the public user, and also meets another 10 conditions as follows. The first point: the open source software allows the free reusable distribution. The license must not restrict that any party sell or give away the software. At the same time, it can't get the sold fees and other fees for this software. The second point: the program must include the full of source code. The license does not allow that getting the source code from any specific forms of the production. The license assures that no one can intentionally to confuse the source code. At

the same time, the users have the right to access to the source code under this license. The third point: which talks about the rights of the derivative work. The license must allow the work-modification and the new-work -derivation . those new's are published under the same license. The fourth point: the integrity of the source code. Licenses and the integrity of permits,which may limit the distribution of the form of the modified source code. The fifth point: license does not discriminate against any specific groups and individuals. The sixth point: license does not limit the use way of any particular field scheme. At the same time, the license can't limit the use way's flexibility and reliability. The seventh point: the distribution of the license. Distribution solutions do not need additional license.The eighth point: the license must not specific to the product. The redistribution of the software does not dependent on the program. The ninth point: license may not restrict other software. This license may not restrict the publishment of the software. The distribute software will be built by using open source. The end point: license rights is neutral. So, it effective limits that the freedom of the code transmission. In other words, it provides the preventive measures.

本文通过一个前提:著作权与合同的对比。作者对开放源代码的定义进行了全面的介绍。开放源码许可的角色，即允许工作许可下的非排他性业务。它不仅意味着源代码被公众用户访问过，而且还满足以下 *10* 个条件源自于 开源协议的摘录: 第一点:开源软件允许免费的可重用发布。许可证不能限制任何一方出售或赠送软件。同时，无法获得该软件的销售费和其他费用。第二点:程序必须包含完整的源代码。许可证不允许从任何特定形式的产品中获取源代码。许可证保证没有人可以故意混淆源代码。同时，用户有权访问本授权下的源代码。第三点:关于衍生作品的权利。许可证必须允许作品修改和新作品派生。这些新版本都是在相同的许可证下发布的。第四点:源代码的完整性。许可证和许可证的完整性，这可能会限制发行形式的修改源代码。第五点:许可证不歧视任何特定的团体和个人。第六点:许可证并不限制任何特定字段方案的使用方式。同时，许可证不能限制使用方式的灵活性和可靠性。第七点:许可证的发放。分发解决方案不需要额外的许可证。第八点:许可证不能特定于产品。软件的重新发布并不依赖于程序。第九点:许可不能限制其他软件。本授权不能限制软件的发布。分发软件将使用开放源码来构建。终点:许可权利是中性的。因此，它有效地限制了代码传输的自由。换句话说，它提供了预防措施。

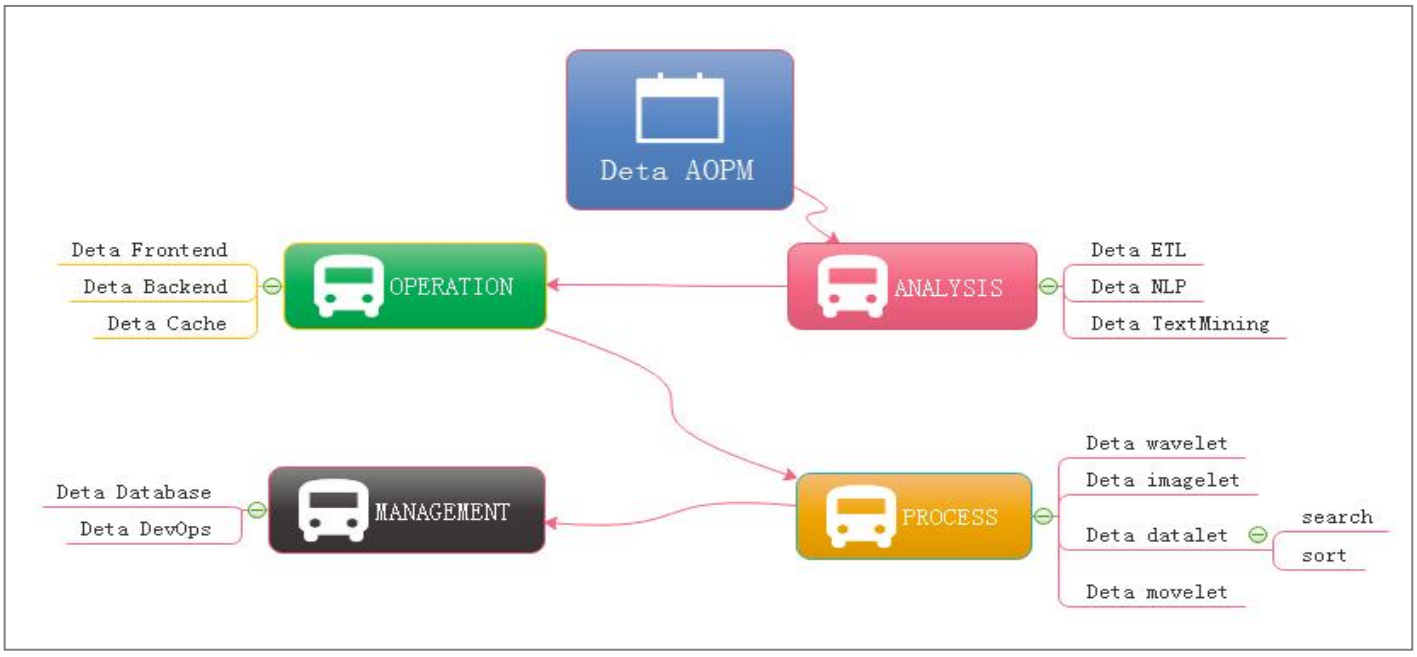


Figure 1-1 AOPM Applications with SDLC

Evolutions

Last year I was asked by so many engineers, almost the same question: how have you make so many projects during the year of 2018? My answer is absolutely: connection. Always, with connection, I got lots of fantasy inspirations on the projects where I undertook. My projects all are lower basic technical factors, with connections, what support me the necessary energy for continuing development on my projects. What means connection? Be an internal union bridge between my projects. For

example Deta NLP and Deta ETL, they both have the same attributes such as AI, Analysis and Data etc, with this connections, my tasks became more dynamically. Everytime before I made a decision of priority levels of my projects, I thought the connection first, Deta projects totally can be separated into three dimensions. Frontend Backend and Storage, as the Figure 1-2, the connection between Deta projects is WEB AI, now is a Bazaar requirement, but we will easy to make estimation of it's future, toward to Cathedral.

去年，很多工程师几乎都是问过我同一个问题:在 *2018* 年，你们是怎么做这么多项目的? 我的答案是绝对的:联系。在我所从事的项目中，通过 *connection*，我得到了很多幻想的灵感。我的项目都是较低的基本技术因素，与连接，支持我必要的精力，继续发展我的项目。连接是什么意思呢? 在我的项目之间做一个内部联系的桥梁。例如 *Deta NLP* 和 *Deta ETL*，他们都有相同的属性，如 *AI*，分析和数据等，通过这种连接，我的任务变得更加动态。每次在我决定项目的优先级之前，我都先考虑连接，*Deta* 项目完全可以分为三个维度。前端后端和存储，如图 *1-2* 所示，*Deta* 项目之间的接是 *WEB AI*，现在是一个市集的需求，但是我们会很容易的估计它的未来，走向大教堂。

Topic: Cathedral and the Bazaar, OSS Book Reading Note

这是我在路德大学研 *3* 写的读后感作业

Cathedral and the Bazaar, this article has a profound implication, the author is a computer scientist with extensive experience. We can say that he is one of the the early code and program contributors in the Unix system. This article describes the Linux development with the revolutionary road, as the process from the bazaar to the cathedral. First, the author tells the contrast between Unix and Linux: now Unix is still popular around the world. Its rigorous structure and contribution to science, let it is proud of the same dignity as a church. Linux looks like a noisy bazaar, the code work in various countries around the world, to solve their own problems and arguing in the forums and communities. Like a bazaar. Then, author points an internal factors to get an in-depth discussion: Unix reason why it has the church's authority, because its development has always been tailor-made by the world's most senior and most eminent researchers and software scientists. Although the discussion, because of the nature of the project-oriented, so that Unix has been applied still to today. Even of the unreasonable original design, through decades of use, engineers have become accustomed to this experience now, there fore, we are called transcendental. which makes Unix feels like a cathedral. The birth of the Linux was different, survival in an all-spittle environment. Every update, are implemented in controversial circumstances. The crowd here, are huge number of scientists, or writers,or code workers or merchants, their common ideal is that make Linux development meets the needs of all groups. Similar a huge bazaar. The author commenced a leno-vo, a conclusion that Linux will eventually beat Unix, Unix gets the range of fresh blood is less than the Linux's, also the number of the Unix team members is less than the Linux's. Unix customers and employees are aging. But Linux development more in line with the user of the needs. Its own development is to establish a relationship on this demand and requirement. Linux is young now. Summary, UNIX and Linux development option is the two kinds of very different road. These processes and methods to determine the fate of the two kinds of software development. Of more optimistic about Linux because it is better adapted to the environment.

大教堂和集市，这篇文章有着深刻的寓意，作者是一位经验丰富的计算机科学家。我们可以说他是 *Unix* 系统中最早的代码和程序贡献者之一。本文描述了 *Linux* 开发的革命之路，从市集到教堂的过程。首先，作者讲述了 *Unix* 和 *Linux* 之间的对比:现在 *Unix* 仍然在世界范围内流行。它严谨的结构和对科学的贡献，让它自豪地拥有与教会同样的尊严。*Linux* 看起来就像一个嘈杂的集市，代码在世界各地的不同国家工作，解决他们自己的问题，并在论坛和社区中争论。像一个集市。然后，作者指出了一个内部因素进行了深入的讨论: *Unix* 为什么拥有教会的权威，因为它的开发一直是由世界上最资深和最杰出的研究人员和软件科学家量身定做的。尽管在讨论中，由于项目的性质是面向的，所以 *Unix* 至今仍被应用。即使是不合理的原始设计，经过几十年的使用，工程师们现在已经习惯了这种经验，因此，我们被称为超越。这让 *Unix* 感觉就像一座大教堂。*Linux* 的诞生是不同的，生存在一片唾沫横流的环境中。每次更新，都是在有争议的情况下执行的。这里的人群，是大量的科学家，或作家，或代码工作者或商人，他们的共同理想是使 *Linux* 开发满足所有群体的需要。类似一个巨大的集市。作者开始了雷诺-沃，结论是 *Linux* 最终会打败 *Unix, Unix* 得到的新鲜血液范围小于 *Linux* 的，而且 *Unix* 团队的成员数量也小于 *Linux* 的，*Unix* 的客户和员工都在老化。但是 *Linux* 的开发更符合用户的需求。它自身的发展就是在这一需求和要求上建立起一种关系。 *Linux* 现在还很年轻。总结，*UNIX* 和

Linux 的开发选择是两种截然不同的道路。这些过程和方法决定了两种软件开发的命运。对 *Linux* 更加乐观，因为它更好地适应了环境。

At figure 1-2, Deta open source main based on AI domain, it already formed as an ecology system, go ahead to the application, thanks.

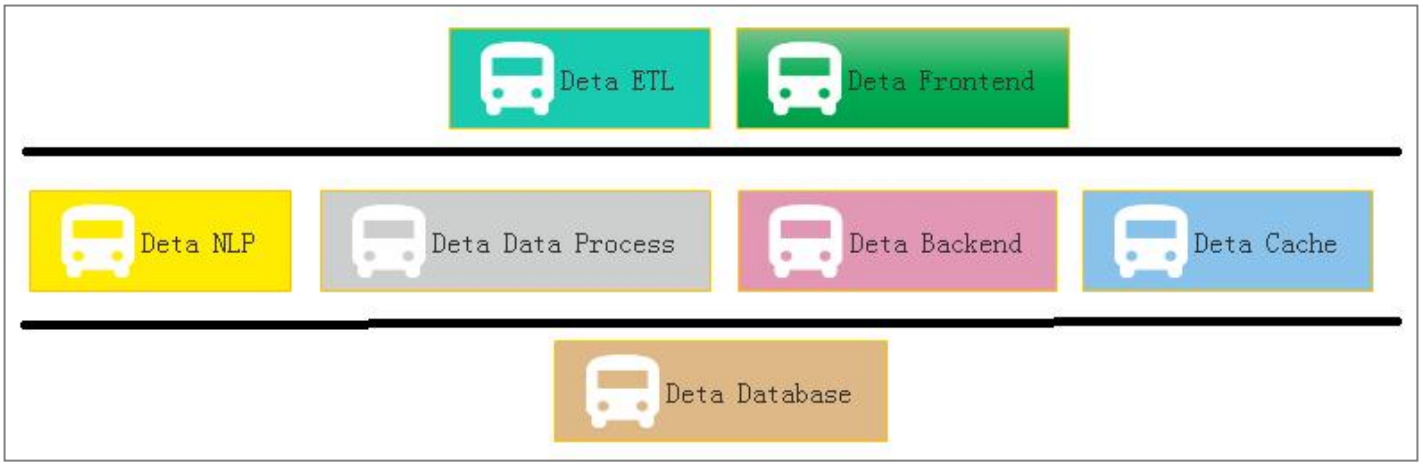


Figure 1-2 Sections of Deta Projects Group

Applications

One question is my friend asked me does Deta support the e-commence logic? Definitly! Please see the Figure 1-3, this is a classic horizontal deployment sample of the real word. Alibaba, Amazon, Ebay and JD etc, all based on this technology, instead of Spring, Deta can be the next generation of technology.

我的一个朋友问我 *Deta* 是否支持电子商务逻辑*?* 当然*!* 请参见图 *1-3*，这是一个经典的服务器横向扩展部署示例。阿里巴巴、亚马逊、*Ebay*、京东等都是基于这种技术，*Deta* 可以取代 *Spring* 成为下一代技术。

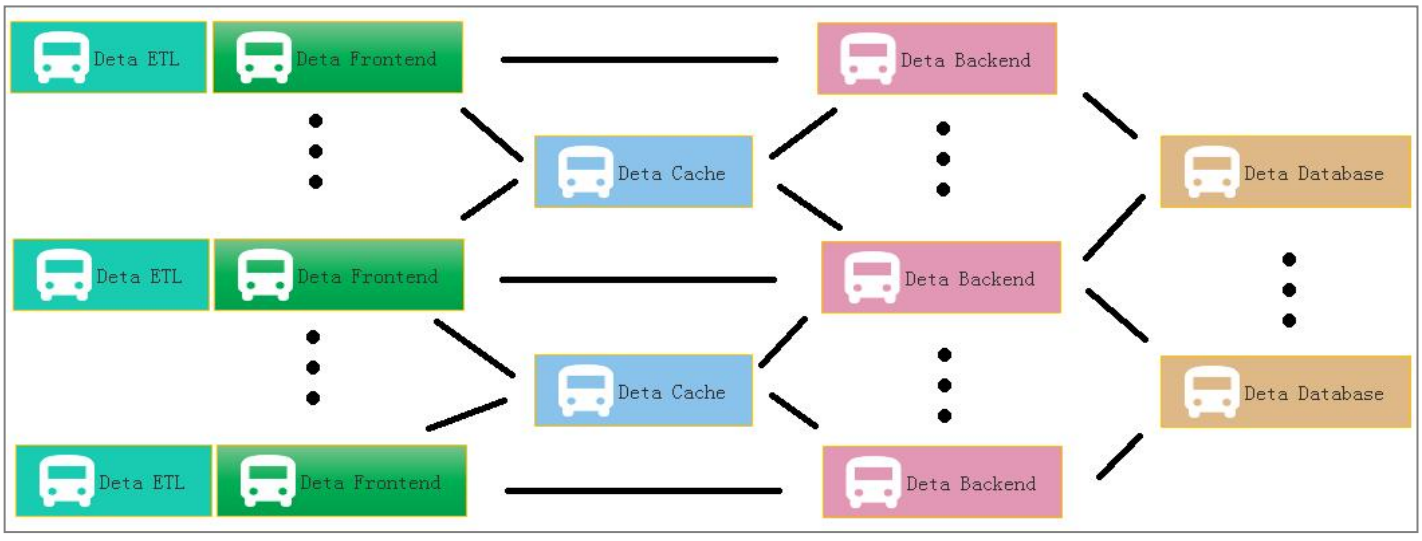


Figure 1-3 Deta WEB Projects System

At Figure 1.4 is a real sample for web Devops by using Deta Open Source.

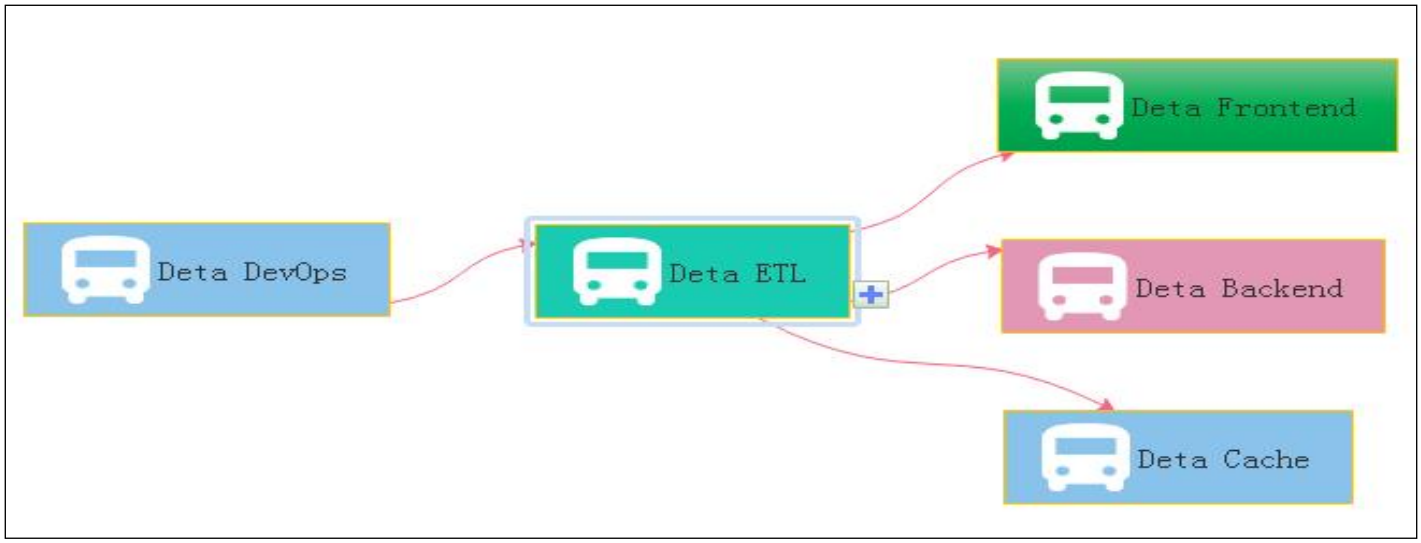


Figure 1-4 Deta DevOps Projects System

II DETA Business backend logic 德塔商业后端逻辑

2010 年以前, 作者在学习的过程中系统的接触了 分析 A, 操作 *O*, 处理 *P*, 管理 *M* 的事物机制, 毕业后有机会在社会上的一些软件公司通过编程来处理这些事物对应的商业业务逻辑. 从法国 *ESIEE* 的 *MP6* 邮件系统研究, 到 上海世博会的 蓝牙群发广告机, 从亚米公司的电商后端计算 到 走四方的全球酒店预订, 作者 一直在思考, 如果前后端系统能像人一样赋予智慧, 多么美好. 于是作者心中的萌芽开始扎根. 有信心设计一套具备类人智慧的架构体系, 满足高速发展的商业智慧应用.

II I DETA Business backend logic history 德塔商业后端逻辑历史

第一次接触前后端分离是在 *2004* 年 作者第一次用 *7week* 平台在浏阳发布网站, 网站是一个 *2* 级域名, 采用第三方服务器, 第一次接触前后端的概念们当时很懵懂. 作者第一次接触 *MVC* 架构是在上海帆腾信息技术公司, 当时的感觉是 *MVC* 能解决各种商业逻辑. 同年作者第一次接触 *MVP* 在做多线程操作蓝牙大文件项目, 感觉也很美好, 觉得 *MVP* 似乎能让架构很好的处理并发计算的问题. *2014- 2017* 年, 作者的工作时间几乎和 *Spring, Martini,* 等多种 *MVC* 架构对应的商业逻辑打交道. 作者认为, *MVC* 赋予智能 迫切.

II II DETA Business backend logic development 德塔商业后端逻辑发展

感谢父亲, *2018* 年对我说按照中医理念设计一个药学辅助搜索软件, 于是开始设计华瑞集医学大数据系统, 当时我觉得 *spring boot, mysql* 都太重了, 如果根据 *CGI* 设计 *socket* 流 数据库 *rest* 握手系统, 那么很多问题都好解决, 于是我开始问题分析, 操作, 处理, 找到了 一些无法替换的基元, 比如 *S* 静态数据, *V* 观测模型, *P*注册方式, *C*控制单元等等, 如果能将这些基于重新设计一套架构, 那该多么的美好. 这里的 *PC* 分离模式, 来自 *2015* 年 *Spring* 的一篇 *IOC* 博士设计论文, 这里表示感谢,
我把 *MV* 综合成 *V* 观测模型, 然后把 *M* 的对应静态数据和函数 *S* 拿出来, 这种 *VPCS* 结构目前满足了我所有的德塔医学大数据应用.

关于 *VPCS* 过多的描述, 我可以摘以前的一段笔记如下: *VPC* 架构编程思想, 经常长年的软件编程，积累了一些对程序实现的思想，通过达尔文进化论的认证，一种有效的 *VPC* 编程理念基于 *MVC+MVP* 的中性耦合文中进行阐述。*V* 是一个观测者模式，类似于存储对象，观测模型。*P* 是处理机，对注册接口的处理。*C* 是控制机，对注册接口的描述和分类。*S* 静态控制机器， 为什么用静态控制机, 优势:*1:* 因为 *PC* 的分离， *c* 模式的函数皆通过抽象的虚函数继承，接口继承接口，接口统一注册，调用极度离散化，达到高速并发迭代的效率。*2:* 实现 *EI* 分离, 跳过 *IOC* 的扫描。*3*： *P* 负责引用和描述， 一个 *c* 可以通过多个 *p* 的描述来实行各种函数运算。映射的控制技术保证线程安全稳定。*4*：*V* 存储各单例类，保证数据冗杂度低，统一回收。

II III DETA Business backend logic application 德塔商业后端逻辑应用

2019 年 一整年, *VPCS* 后端引擎 逐渐形成一些规范化函数和论文, 应用在德塔 前端, 后端, 缓存, 数据库, 等子系统中, 我对他们的评价是, 轻巧, 可扩展. *VPCS* 逐渐集成在 养疗经 和 华瑞集作品中. 当然不足也很多, 最大的不足是没有做到自我修复, 虽然我设计了 *sleeper* 和 *hallkeeper* 机制, 但是这些机制只是 我通过决策树 来 完成相应的业务逻辑单元,不

是类人的进化思维. 至少我觉得不是类人智慧，确切的说目前只是人工智慧. 一种 *AOPM VPCS* 对应的人工智能逻辑. 而不是我想要的类人进化智慧逻辑. 于是我又开始了类人计算探索. 关于 *VPCS* 应用原理描述: 作者设计了一篇论文如下:

VPCS Backend Theory And Its Application

Mr. Yaoguang. Luo

Liu Yang Deta Software Development Limited Company, Hunan, China,

313699483@qq.com

Outline: *due to the development of the software acquisition and definition in what we use the code theory always in messy and unforeseeable status. A new method of the coding style like VPCS that will show in this topic paper, feel free to resonate with my imagination of the portrait—VPCS(Vision, Process, Controller, Sets) theory, fun yet? Not only this paper will gazer a big point how we show the constructions of the VPCS, you guys also sure to get lots of idyllic landscapes of the coding sections. While you got lots of the illness codes at the so messy fungus projects, I guess at this paper out where you are finding anxiously. Let's catch more opportunity about how does the VPCS working, executing and scheduling in our software project and make the software fast, fast and safe! lets go, So the key words as below:*

大纲:由于软件的发展，在获取和定义方面我们所使用的代码理论总是处于混乱和不可预见的状态。一种新的编码风格的方法，如 VPCS，将在这篇专题论文中展示，请随意与我想象的肖像- VPCS(视觉，操作过程，控制处理器，静态集)理论产生共鸣，有趣吗?不仅这篇论文将关注一个大的点，我们如何显示构建的 vpc，你也一定会得到许多田园风景般的编码部分。当你在如此混乱的细琐项目中得到许多病垢代码时，我猜在这篇论文中你会感到不安。让我们抓住更多的机会，如何工作的 vpc，执行和计划，在我们的软件项目，使软件快速，快速和安全!let go，所以关键字如下:

Quantum Sets, Concurrent Consumer, Vision, Scheduler, Threads, Surf.

Introductions

Let see the verbal keys, the first time you …, okay, get any sense? Sure, this paper is not talking about the human careers, truly about SOFTWARER, as a human, if you got my points, yes, cool! Make any sense? Let's see the landscape as below figure 1-1.

让我们看看语言键，第一次你…，明白了吗?当然，这篇文章并不是在讨论人类的酒店职业生涯，而是真正地讨论作为一个类人的软件生命存在，如果你明白我的观点，是的，很酷!任何意义?让我们看看如下图 *1-1* 所示的景观。

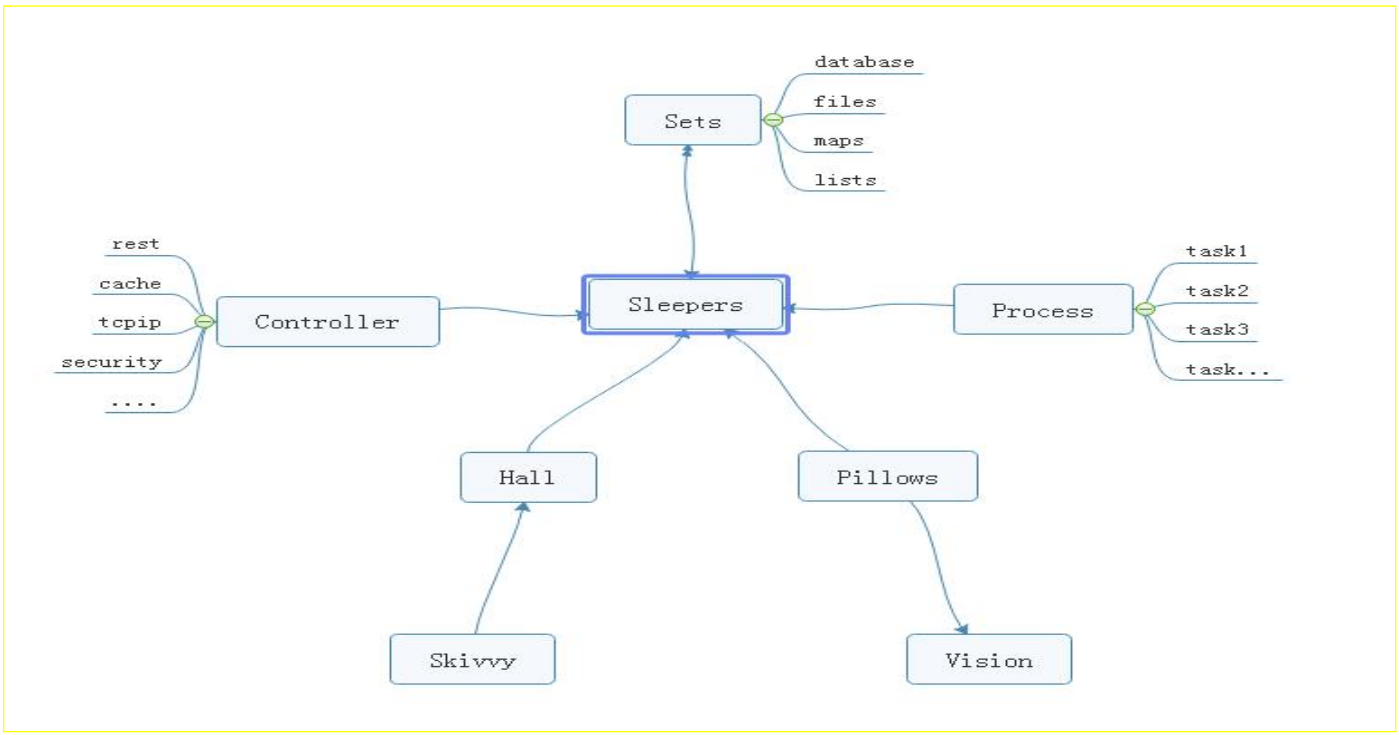


Figure 1-1 VPCS STAR MODEL

From the ordinary software development architecture, always like a factory model, for instance, controller, transaction delegate, web service, job bean, data DAO, like that of traditional backend or front end coding style, but, compare now the seamless clients services system, those model more and more not suitable for us for the project application, at least in the light level, multitasks, satellite boots projects system, if we choice the factory model, you will feel so heavy. But the big conflict problem is where the factory model was used in all and all bazaar companies. Even more CTOS that I met before often complaining about the reference room likes that "we need one server for database system, one more for cache system, one more for front end, one more, for backend, one more…."after that what do you think? My lord…

从普通的软件开发架构来看，总是像一个工厂模型，例如，控制器，事务委托，*web* 服务，工作 *bean* 数据通道,像传统的后端或前端编码风格,但是，对比现在,无缝的客户服务系统,这些模型越来越不适合我们项目的应用程序中,至少在轻微水平级别，多线程事务的同时，卫星集群般部署的项目系统，如果我们选择工厂模型，你会觉得很沉重。但最大的冲突问题是，工厂模式被用于所有的集市公司。更多的 *cto* 经常抱怨资料室，比如"我们需要一个数据库服务器，一个缓存服务器，一个前端服务器，一个后端服务器，一个……"那之后你觉得怎么样?"我的主...

Finding a new method of how to integrate the sets about the micro satellites service in the same sever, and make them small, lightly and faster for the commence service, now become a fatal topic. Which can be a pretty warm-up for where I make an explanation for VPCS. The VPCS model, only includes four aspects. Vision, Process, Controller, Sets, and those factors makes an interactions in the sleeper containers. Let talk about the definition of the sleepers. From the software engineering domain, the sleepers are more like an identified thread person. Who can make a lot of fantasy dream in a Hall, what means a dream? Dream is a requirement what the consumer really needs to finished. But here the dream can be separated out more tasks, those tasks will register the ID in the Pillow, so that the sleeper hugs the pillow then goes into the hall and make a dream. Got an idea? Cool. So what does the sleeper does in a hall? The answer is to make all kinds of the dream. For example if we want to build the web service to get rest call, and return the JSON feedbacks, we only need to do like the way: Firth, build rest call path in the controller; Second: register the call requirements as a dream; Third, build the sets of the dream in the pillow, Fourth hire a sleeper to hug this pillow, and go to the hall to make a dream process. At last but least: return the dream goods. Any sense? Cool! For this unique instance, you will know that the sleeper was more like a socket, and the hall more like a thread pool, the pillows like the single vision instance, and the sets like a vision storage, the controller and the process those two sections is a common way of the factory model. The steps landscape of the sleeper who makes a dream as bellow figure 1-2.

如何在同一服务器上集成微卫星蜂群服务设备，使其小型化、轻量化、快速化，成为当前重要的课题。这对于我解释 *vpc* 来说是一个很好的热身。*VPCS* 模型，只包括四个方面。视觉、过程、控制器、设置以及这些因素在轨枕容器中进行交互。让我们来谈谈梦想家的定义。从软件工程领域来看，"梦想家休眠者"更像是一个已确定的线程人员。谁能在大厅里做很多幻想的梦，梦是什么意思? 梦想是消费者真正需要完成的一项要求。但是在这里，梦可以被分离出更多的任务，这些任务会在枕头上登记 *ID*，所以睡觉的人拥抱枕头，然后走进大厅做一个梦。有一个主意吗? 酷。那么卧铺在大厅里做什么呢? 答案是做各种各样的梦。例如，如果我们想要构建 *web* 服务来获取 *rest* 调用，并返回 *JSON* 反馈，我们只需要像这样做: 首先，在控制器中构建 *rest* 调用路径; 第二: 把呼叫要求登记为一个梦; 第三，在枕头上搭建梦境的布景，第四，雇一个睡觉的人来抱这个枕头，然后去大厅做一个梦的过程。最后，但最重要的是:把梦想实现的结果输出。任何意义? 太酷了! 对于这个独特的实例，您将知道睡眠更像是一个套接字，和大厅里更像是一个线程池，枕头像单视觉实例，像是和集存储、控制器和过程这两个部分工厂的模型是一种常见的方式。做梦的人的台阶景观如图 *1-2* 所示。

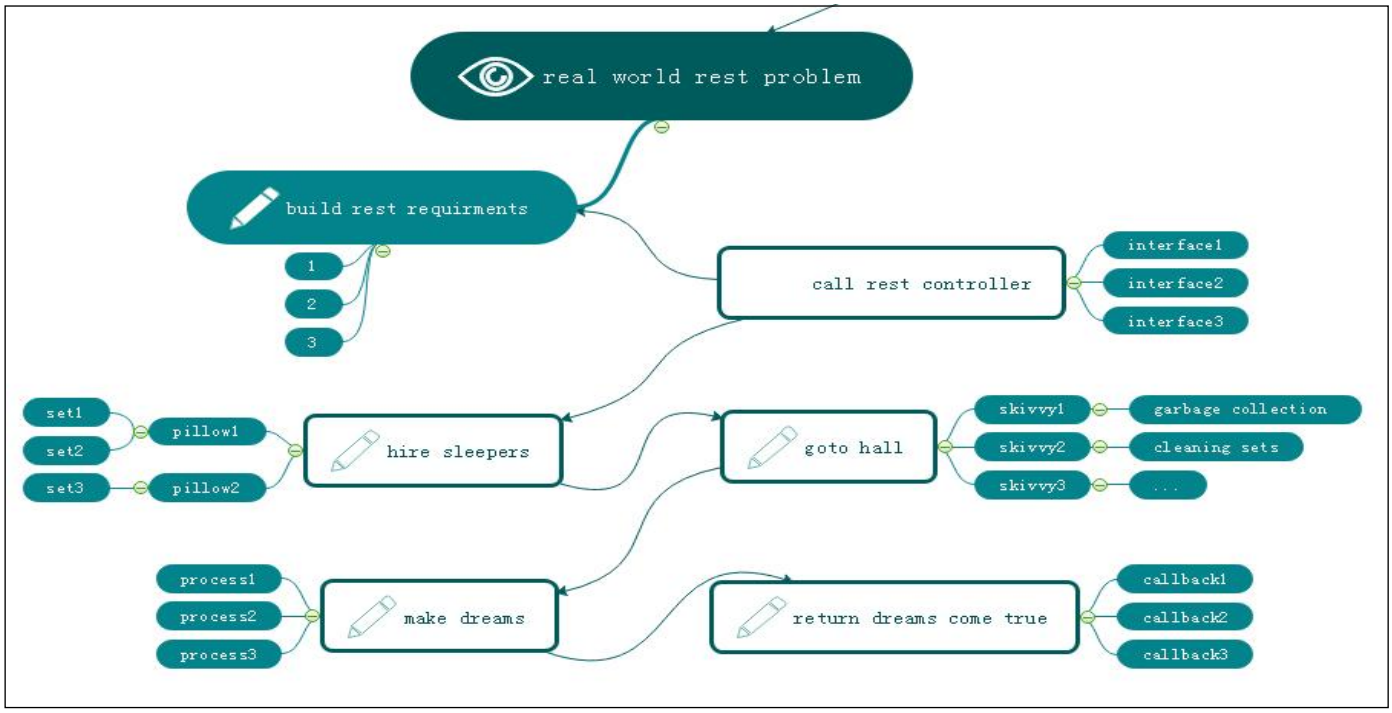


Figure 1-2 VPCS BACKEND MODEL

Focus on this landscape, mostly different to the MVC: Model View Controller, MVP: Model View Presenter or other architectures we know before. but is very easy to understand after you read for a while. Too simple. Sleeper makes dreams come true, hall container sleepers, skivvy make up the hall, pillow clear and wake up the sleepers who often lost in finding the way in the dream. Got fun here, but I would hear more argue voice details of my VPCS, desktop App once said: VPCS is good in the concurrent WEB project, but not suitable for the desktop applications. Ok, follow this question, let make a new landscape based on desktop application as below figure 1-3.

专注于这个景观，主要不同于 *MVC: Model View Controller, MVP: Model View Presenter* 或我们之前知道的其他架构。但是读一段时间后就很容易理解了。太简单了。睡眠者使梦想成真，大厅集装箱睡眠者，*skivvy* 使大厅，枕头清楚，唤醒经常在梦中迷路的睡眠者。在这里得到了乐趣，但我将听到更多的争论声音细节关于我的 *vpc* 桌面应用，曾经有人说:*vpc* 是好的并行 *WEB* 项目，但不适合桌面应用。好的，按照这个问题，让我们创建一个新的基于桌面应用程序的横屏，如下图 *1-3* 所示。

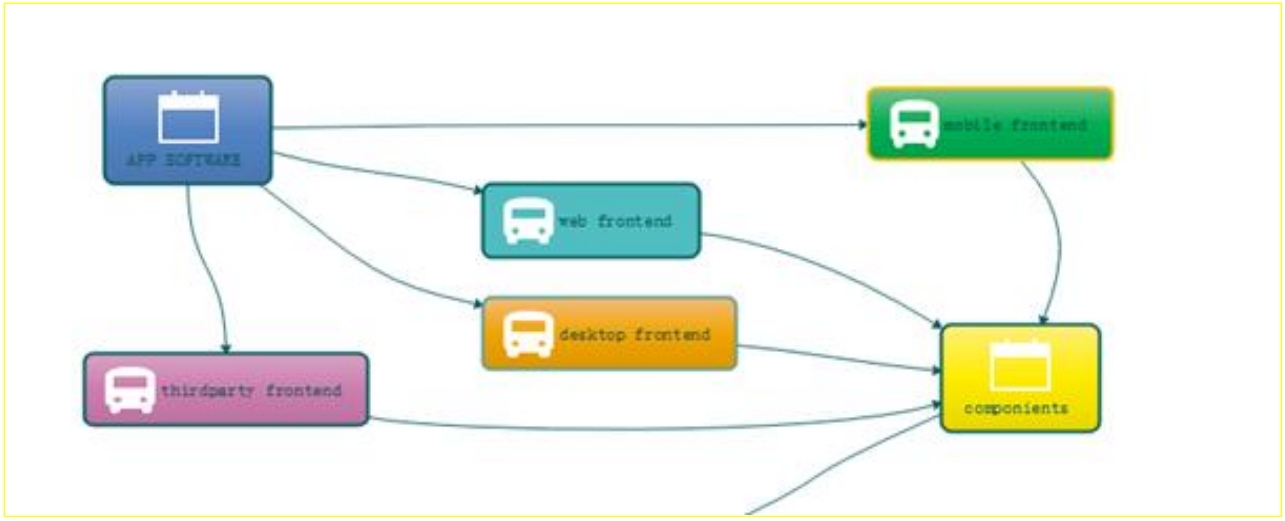


Figure 1-3 VPCS WORK WITH FRONTEND

From this picture, we know all of the software can be fast and safe while using VPCS, because it is already separated out the big system into backend and frontend two parts. and VPCS keeps safe and fast in the backend section. Compare to the MVC, VPCS will get more cautious and details, and compare to MVP, VPCS also will get more safe and high efficiency. Those factors are why I will make inauguration here. In the common software engineering cycle life times, scientist used to build front end and backend for all kinds of the software applications, because it is easy to control. Why? Frontend only spend time to make design, and Backend for the data operations. Using VPCS system, we don't care about what they do for the front end, we only fit about what they want. A lignment that gets a blame and fix, then return OK, the restful service developer makes a voice that http functions are concurrent functions. At here, VPCS will say: concurrent functions are safe functions. We guess in the future REST-VPCS will be used in multiple WEB service. Especially in the high speed, efficiency, micro web systems with high level security for example medicine, DNA, cloud server, electronic police system and ecommerce systems etc.

从这个图中，我们知道所有的软件在使用 *vpc* 时都是快速和安全的，因为它已经把大系统分成后端和前端两部分。*VPCS* 在后端部分保持安全和快速。与 *MVC* 相比，*VPCS* 会更加谨慎和细致，与 *MVP* 相比，*VPCS* 也会更加安全高效。这些因素就是我在这里就职的原因。在常见的软件工程周期生命周期中，科学家常常为各种软件应用程序构建前端和后端，因为它易于控制。为什么？前端只花时间做设计，后端只花时间做数据操作。使用 *vpc* 系统，我们不关心他们为前端做什么，我们只适合他们想要的。得到责备和修正，然后返回 *OK, restful* 服务开发人员发出声音说 *http* 函数是并发函数。在这里，*vpc* 会说*:* 并发函数是安全函数。我们猜测将来 *rest - vpc* 将被用于多个 *WEB* 服务中。特别是高速、高效、高安全级别的微 *web* 系统，如医药、*DNA*、云服务器、电子警察系统、电子商务系统等。

最近养疗经引擎也开始基于 *vpcs* 进行布局. 其子系统已全部采用 *vpcs* 结构.

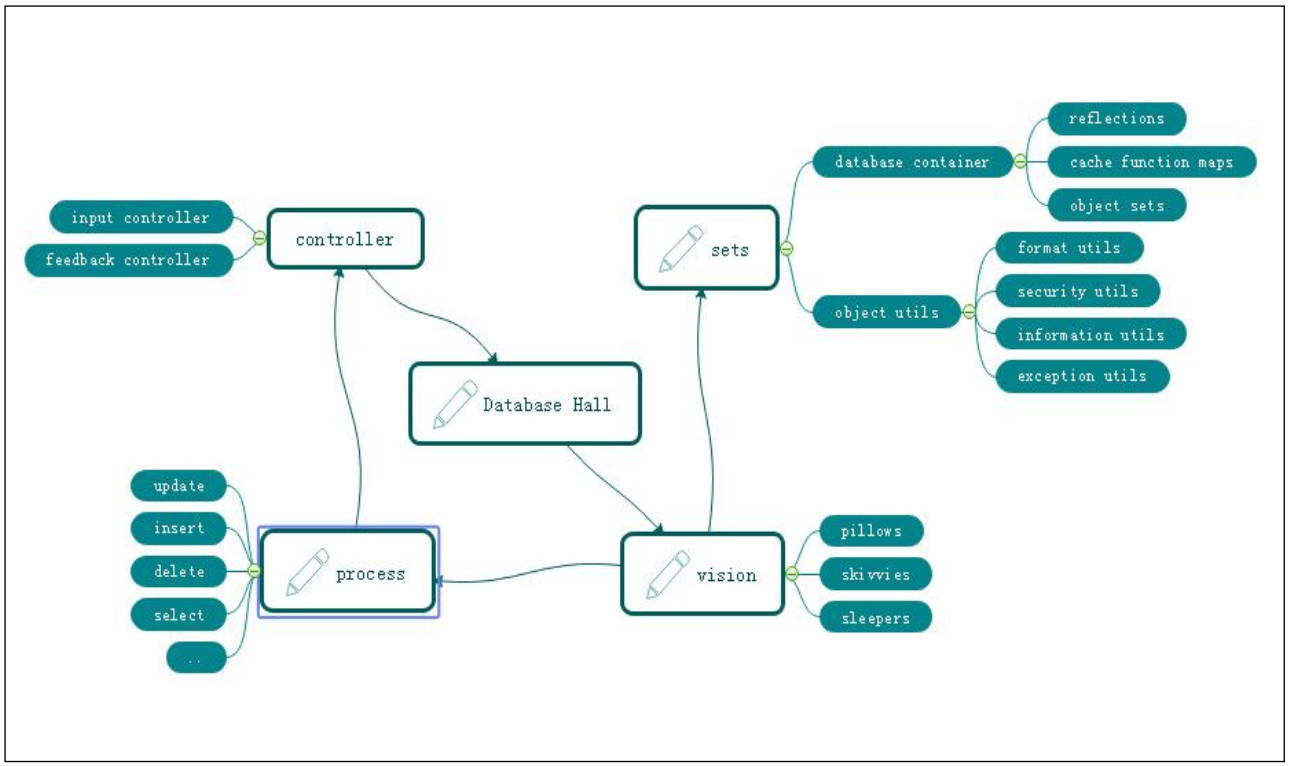


Figure 1-4 VPCS FOR DATABASE SYSTEM

As the figure 1-4, a new method of the Database system designing shows us VPCS is a pretty way for the modern data information management system. Definitely used in the DETA Database system. For this instance, do we get a view that the controller section of the factory model becomes thin yet? Controller only works for the hands transactions, for example that the controller get an input requirement such as select SQL, then immediately call the hall keeper to register this SQL and hire new sleepers to make a result. Because of the VPCS. Once it happened any exceptions, will very easy to awake sleeper and let them get theirs working papers out, finally call skivvy to fork the sets to the fresh sleeper. This method mostly be like a Count

Down Latch model, once the sleeper gets the dreams come true, then told the hall keeper for the feedback, hall keeper will makes a type procession to return after everything goes well, This method mostly be like a Cyclic Barrier model.

如图 *1-4* 所示，一种新的数据库系统设计方法向我们展示了 *VPCS* 是现代数据信息管理系统的一种很好的方式。明确用于 *DETA* 数据库系统。对于这个实例，我们是否得到了工厂模型的控制器部分变薄的视图？控制器仅适用于 *hands* 事务，例如，控制器获得一个输入需求*(*如 *select SQL)*，然后立即调用大厅管理员注册这个 *SQL* 并雇佣新的休眠人员来生成结果。因为 *vpc*。一旦发生任何异常，会很容易叫醒睡眠者，让他们拿出自己的工作底稿，最后叫 *skivvy* 把餐具叉给新的睡眠者。这种方法大多类似于一个倒计时锁存模型，一旦睡眠者的梦境实现，然后告知大厅管理员进行反馈，一切顺利后，大厅管理员会做出一个类型的队列返回，这种方法大多类似于循环障碍模型。

Questions

How does skivvy doing? please see the figure 1-5 the hall building need a singler instance like a home keeper but here is a hall keeper, any else, this person is very important for keeping the VPCS safe, because all of the skivvies will be managed by him. You will see, the memory check, JVM garbage collection, disk cleaning, thread status management, deadlock alarm, security protocol all and all in one at here. Mostly like a static class in the VPCS system. If we need to know every thing about skivvy's work status, ok just call the hall keeper.

skivvy 做了些什么事务？请见图 *1-5*，大厅建筑需要一个像管家一样的单例，但是这里有一个管家，其他的，这个人对于保持 *VPCS* 的安全非常重要，因为所有的 *skivvies* 都将由他管理。您将在这里看到，内存检查、*JVM* 垃圾收集、磁盘清理、线程状态管理、死锁警报、安全协议，所有这些都集于一身。大部分类似于 *VPCS* 系统中的静态类。如果我们需要知道 *skivvy* 的工作情况，可以打电话给大堂管理员。

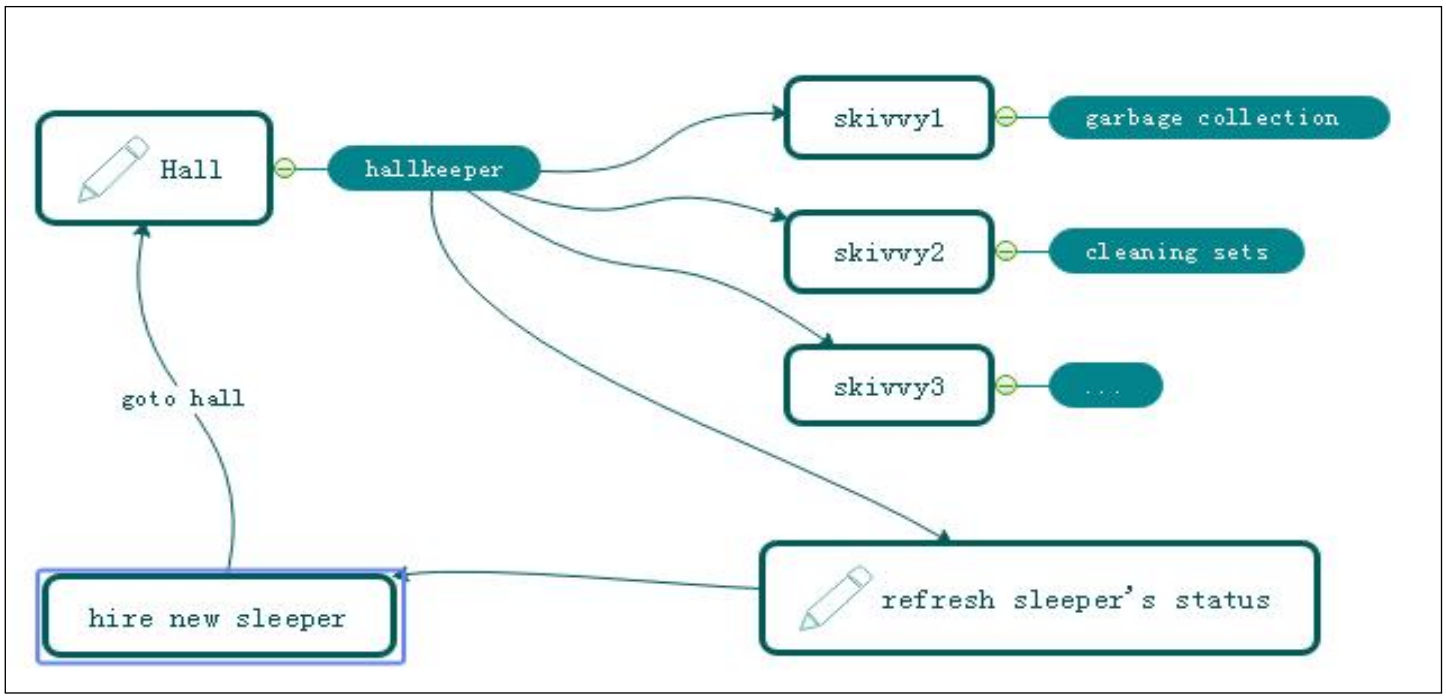


Figure 1-5 VPCS KERNEL

How does sleeper doing? Make a dream? Cool, you shoot!, in the VPCS system, it doesn't have the definition of the process, everything likes subsets. Immutable or unlined, hall keeper get request from visionary and hire the sleeper, who is likes a thread, get requirement, add those sets in pillow, hugs pillow then go to hall to make a dream, after that then return the callback to hall keeper what they did. Fun yet?

睡觉的人怎么样?做一个梦吗？酷,你正中目的*!* 在 *VPCS* 系统中，它没有进程的定义，一切都像是子集。不可变的或无线条的，厅管得到梦想者的请求，雇佣睡眠者，谁是线程，得到要求，添加那些套在枕头，拥抱枕头然后去厅做一个梦，然后返回回厅管他们做了什么。有趣吗？

What does the sets meaning? Sets, is a format of the data where appearing in the VPCS system. For the static prototype, it used like a concurrent hash table, and list which can be copy base on writing format, the single instance, it always runs in the static function or be liking an interface implementation because need safe at the same time, so that compare to the factory model, it is too simple and without annotation. Everything becomes easy in this environment.

这些集合是什么意思？是出现在 *VPCS* 系统中的数据的一种格式。静态原型，它像一个并发的哈希表，使用这些列表，可以复制基本写作格式，单一实例，它总是运行在静态函数或像是一个接口实现,因为需要安全的同时,与工厂模型相比，它太简单，没有注释。在这种环境下，一切都变得容易了。

The one more question is that so many peoples asked me what does the sequence diagram of the VPCS, because they really want to know why VPCS is faster and safe. Ok, please see the figure 1-6, the answer is absolutely, VPCS main components of the time sequence only contains five aspects. Almost similar like the hotel management. Certainly, we are talking about VPCS software, not for guesthouse. You will see that the rest call only makes the interactions with the hall keeper. And hall keeper got two jobs, one for waiting the fresh sleeper and one more for giving task to skivvy. The sleeper only hugs the relate pillow and make the dreams come true. Fun yet? Cool. VPCS only take cares about how does the sleeper's imagination and skivvy's working status. If is the pillow broken? Make new pillow, got lazy sleeper? Get out his working papers, got a cheat skivvy? Fix of fire him, the real source of the java version project for the VPCS only 30kb, we will find more sources or documents from the reference links at the end.

还有一个问题是，很多人问我 *vpc* 的序列图是什么，因为他们真的想知道为什么 *vpc* 更快、更安全。好的，请看图 *1-6*，答案是绝对的，*VPCS* 主要组件的时间序列只包含五个方面。和酒店管理差不多。当然，我们说的是 *vpc* 软件，而不是宾馆用的。您将看到 *rest* 调用只与大厅管理员进行交互。厅长有两份工作，一份是等新来的人，另一份是给佣人分派任务。睡觉的人只会抱着枕边的枕头，让美梦成真。有趣吗？酷。*VPCS* 只关心睡眠者的想象力和 *skivvy* 的工作状态。如果枕头被打破了？做新枕头，遇到睡懒觉的梦想家吗？把他的工作文件拿出来，有个服务骗子吗？修复或者炒了他，真正的 *java* 版本项目的源代码为 *VPCS* 只有 *30kb*，我们将在后面的参考链接中找到更多的源代码或文档。

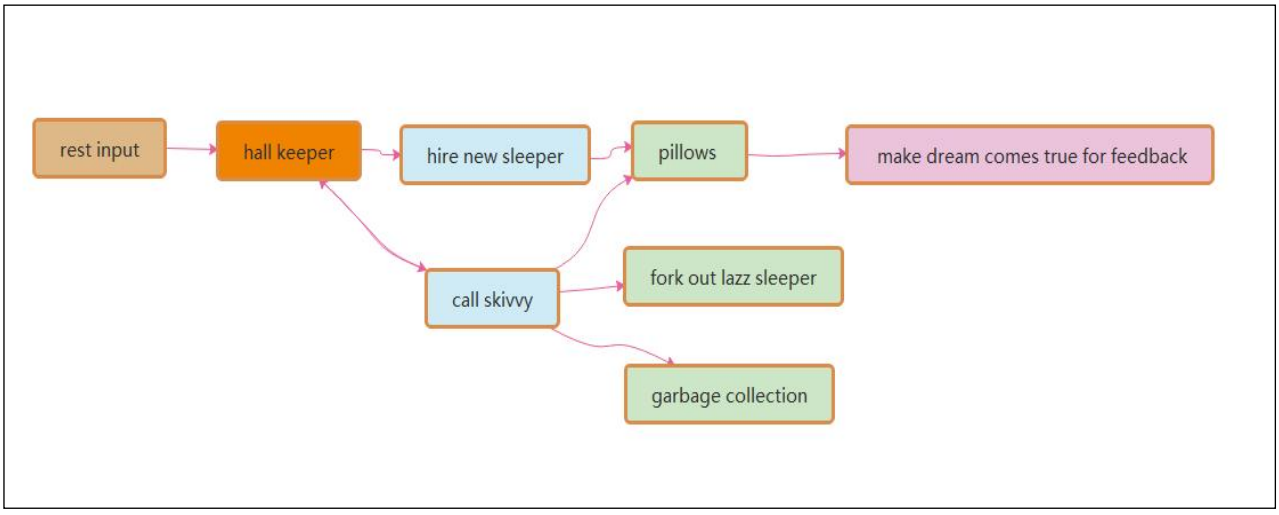


Figure 1-6 VPCS Sequence Diagram

I always be asked by the colleagues that once said: how does the hall build? I answered them, such like the hospital, no one cares about the address of hospital, because they just call the cell phone number when will get a directly feedback. This is why I need a hall keeper role in the gate way. For the instance about figure 1.6.1, this sample is a true demo in the real world for the WEB rest service. Its very important to create a player role such like hall keeper. what would likes about author's theory? Because of the maintenance. Because of whom, the software build team are very easy to make a maintenance web portal, all of the system current status will be solved on this html page by DEVOPS.

我总是被问道 *VPCS* 中的 大堂经理用来干什么，我回答说 正如一个医院，没人关心医院的地址，因为他们拨打

120 救护车就来了. 大堂经理就像这个接电话的. 后端接口请求同样, 没有哪个事务逻辑需要知道怎么被执行,因为大堂经理按照索引规范已经登记好了. 这种登记方式非常容易被开发运维人员维护操作.

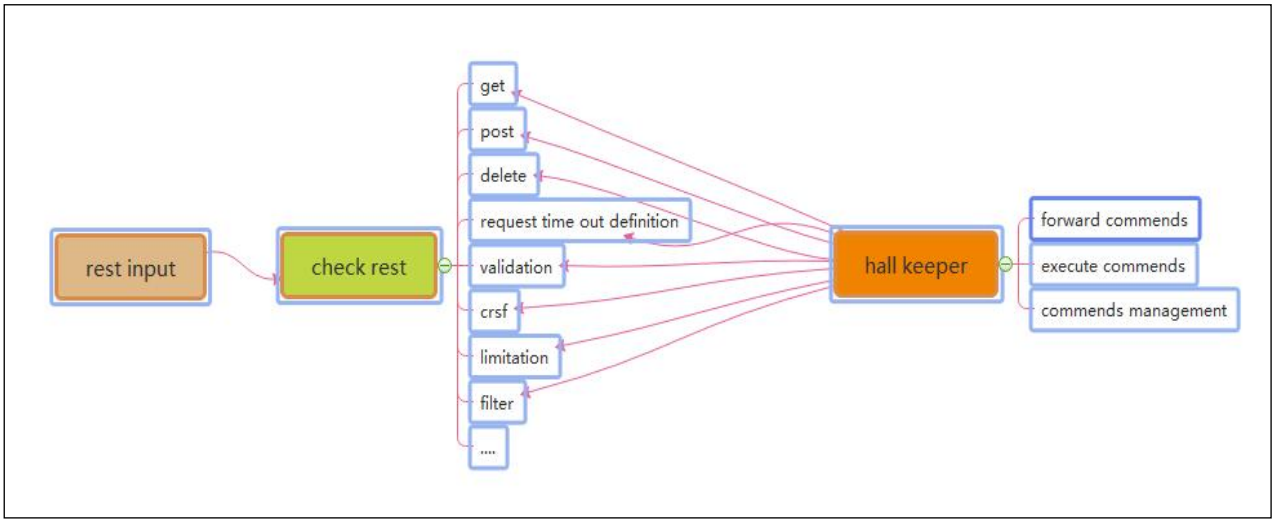


Figure 1-6-1 The Interaction Between Rest Call and Hall Keeper

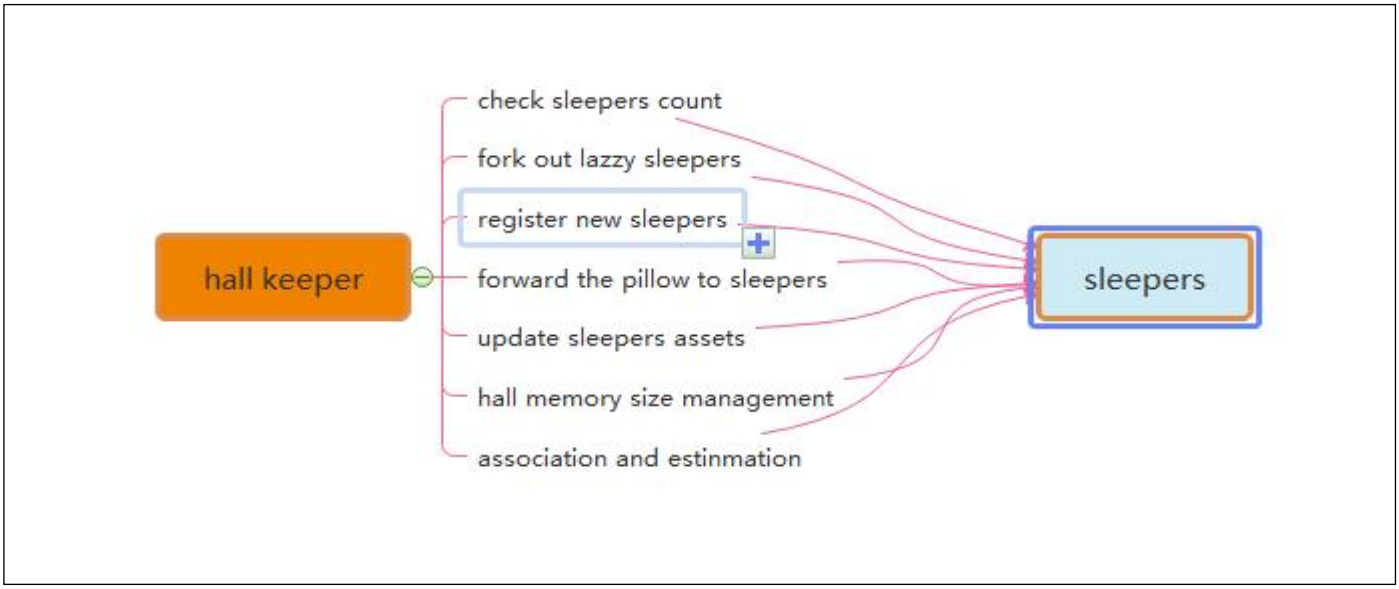


Figure 1-6-2 The Interaction Between Hall Keeper and Sleepers

Many of these software developer also asked me how and why we fork out the LAZZY sleepers excluding their sets. Arthur answered because of the pillows. When the sleepers be hired from the hall keeper, they will get an independently pillows such like static functions. So that sleeper only has their own indentify attributes and unique information as the singe instance class. Once they got theirs working paper, the pillows they used will be arranged to the new fresh sleeper, this theory keeps safe, quality and quantity. Like figure 1.6.2.1 VPCS kernel

很多软件开发者同样会问我 如果删除掉一些睡懒觉的线程, 他们的枕头怎么处理? 这就是 *vpcs* 的闪光点, 因为枕头是 *id* 标识的静态数据, 所以, 是可以重用的. 新的梦想家可以从这些枕头里提取有用的数据碎片. *(*当然这里是软件不是酒店,酒店的枕头是要高温清晰消毒的^-^*)*

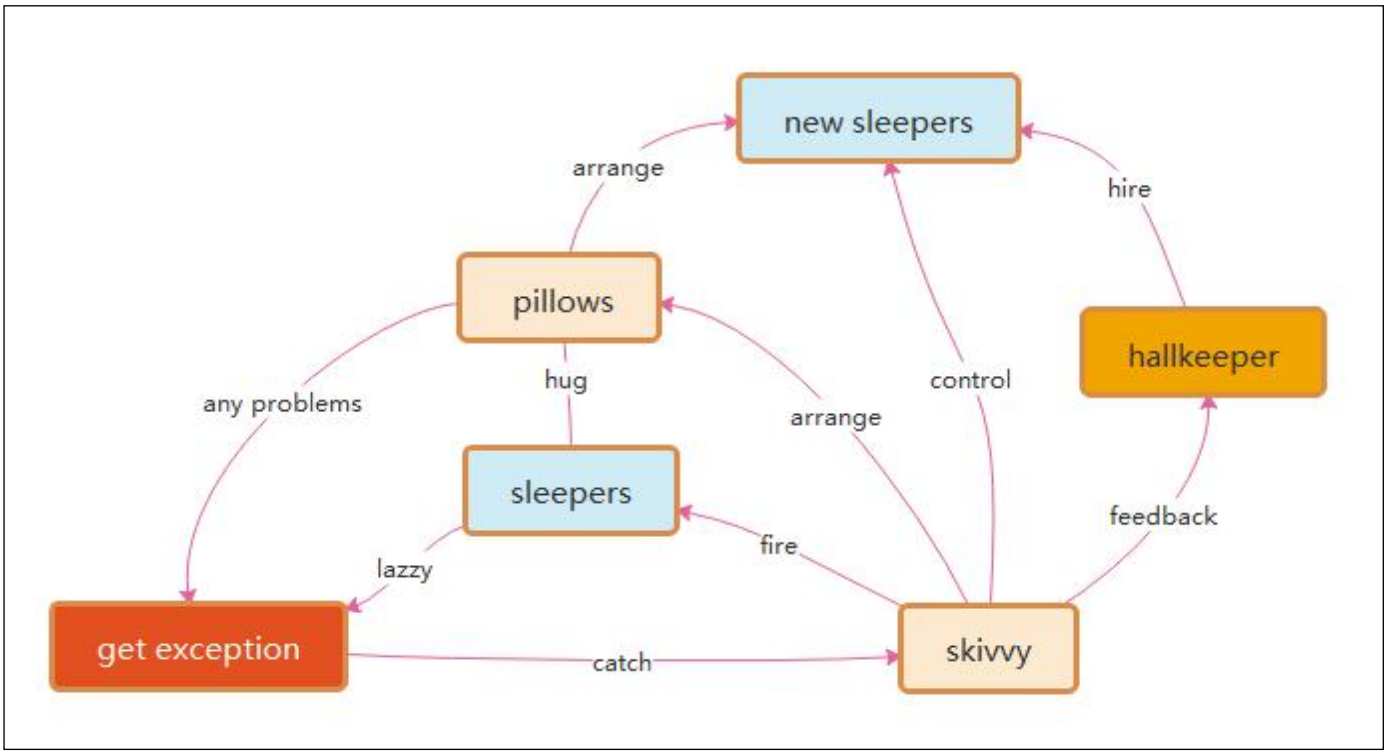


Figure 1-6-2-1 VPCS kernel

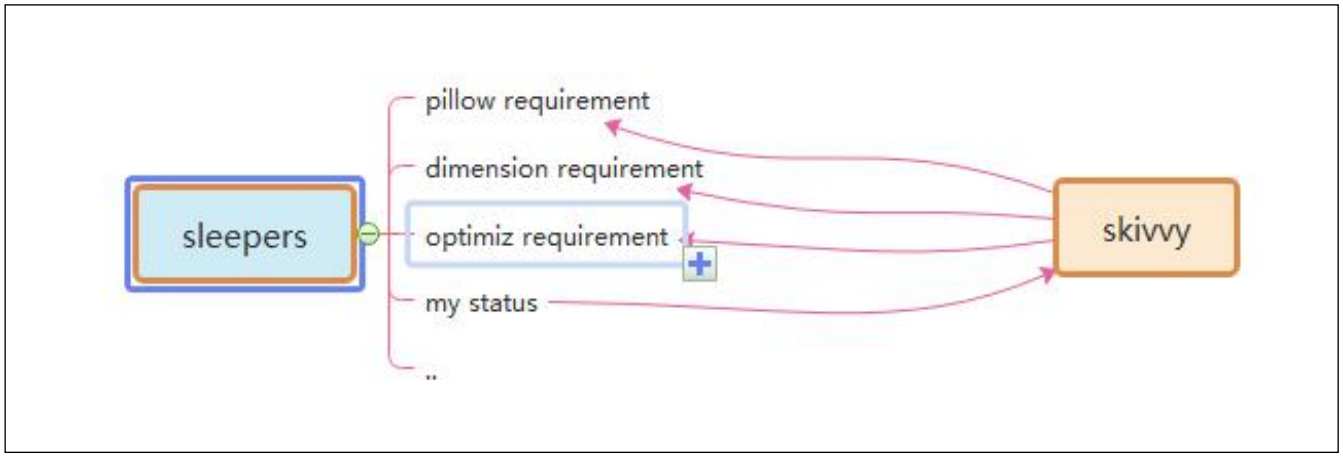


Figure 1-6-3 The Interaction Between Sleepers and skivvy

Many times, I got questions about the DEVOPS, they really worry about VPCS if suitable for their project system maintenance? The answer is absolutely, as the Mr. Ray 274138705@qq.com once said: we are DEVOPS, at least we need three important keys in our environment assignments: ***implementation capacity, transparency and maneuverability****. How does the VPCS supports us for daily works? Because of Hall Keeper and skivvy, as figure 1-6-3 and 1-6-4. DEVOPS will get all of these transparency information about project from hall keeper under the encryption and security issues. Also, hall keeper will directly get the rule for DEVOPS by rest calls, then makes to commend to skivvy. All of the information and record logger will be cached by hall keeper, that keeps controllability. The html control page will make an interaction between hall keeper and DEVEOPS, which keeps safe, implementation capacity, transparency and maneuverability. This is my true answer.*

很多时候，我被一些运维问道,采用了 *vpcs* 架构怎么进行系统维护？ 答案是很明确的,*ray* 先生曾问我 作为运维他们往往纠结于 耦合度，透明度和可维护性 *vpcs* 具体是怎么体现的？我的回答是 大堂经理与服务员的命令落实，这些落实情况可以进行完整的可观测的日志记录，内存状态，中间属性标识.,

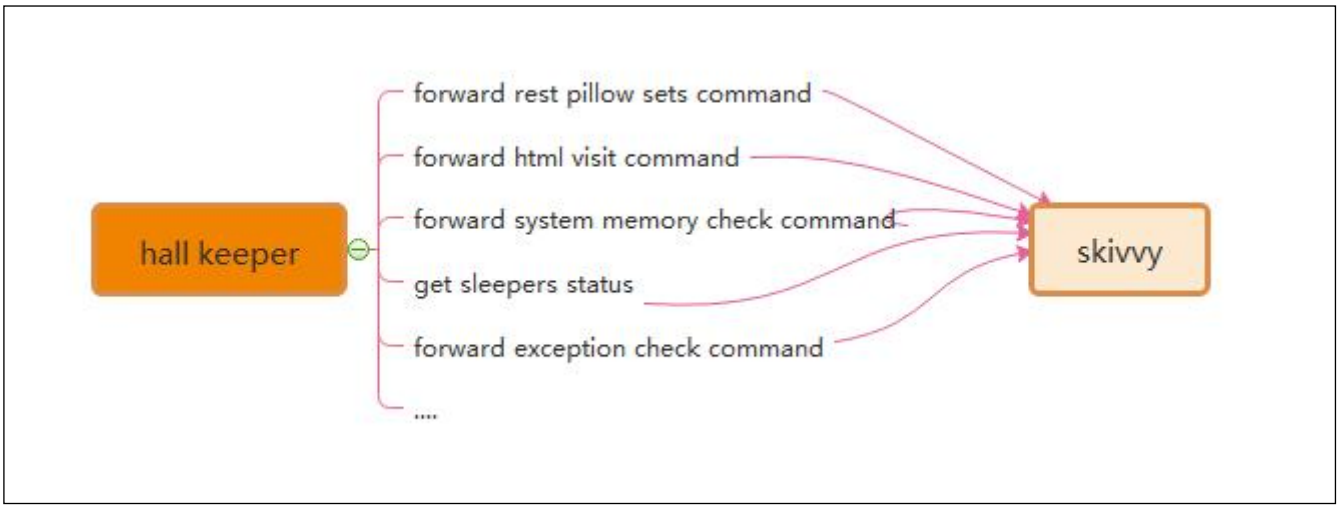


Figure 1-6-4 The Interaction Between Skivvy and Hall Keeper

Recently Mr. Yang 1291244774@qq.com who asked me about VPCS of IOS desktop APP, where and how to avoid the data leakage risks. Because he really worries about the separation between controller and process. Following this topic, my answer that the key is the separation between pillows and sleepers. Due to the pillows all have their own unique ID, skivvy will easily arrange the pillow to new sleeper after the original sleeper who made problems. Make unique ID and arrangement by ID, is the key method. Also for the rest call service, the asymmetrically irreversible combination encryption is one of the best solutions to the data leakage controller. VPCS seems so smart.

最近 *yang* 问我 *vpcs* 能用在 *IOS* 中么? 因为它担心 *pc* 分离 有可能 导致 数据的溢出, 我的回答是绝对的安全, 因为每一个线程都进行了唯一的公有标识, 连静态属性, 单例属性都有明确的属性表示, 这个标识能在内存中时时刻刻着找到它的状态. 另外 *rest* 的异步消息握手同样能够进行精准的维护.

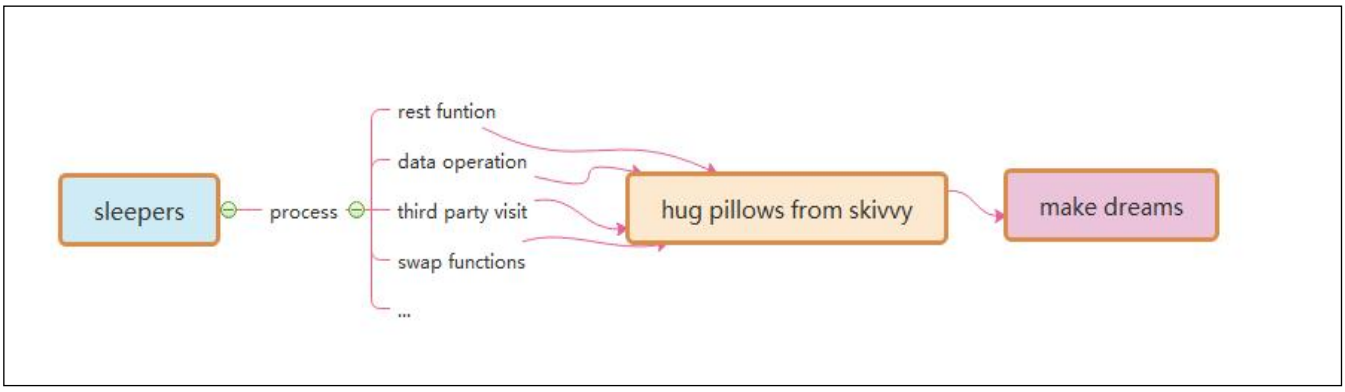


Figure 1-6-5 The Interaction Between Sleepers and Pillows

III DETA Catalytic computing 德塔催化计算

2019 年我迷茫了一个月, 类人计算, 怎么实现? 困扰了我很久, 怎么开始? 我没有头绪, 我于是开始翻阅我这 *20* 年做的笔记, 有了! 做做基础研究! 根据我的笔记来挖掘一些未知的基础知识. 我非常的快乐, 因为我有 *2014* 年的左右比对法快速排序的成果, 快速傅里叶的蝶形计算原稿, 华瑞集中文分词的作品, *UNICORN ETL*, *socket* 流 *PLSQL* 数据库 等, 一个想法应运而生, 对 持续专注的优化它们, 不断的重复的细化, 优化, 测试, 记住这些优化的思想, 这种思想,我思考了很久, 一定是类人的进化思想.

III I DETA Catalytic computing history 德塔催化计算历史

既然想到了闪光点,那么就应该需按照操作方法. 德塔的第一篇催化计算的细化方式首先在德塔分词中,逐渐体现, 比如语义词性分析的细化, 流水阀门的优化, 离散数据的非有理条件变换, 同频算子的过滤, 计算高峰的过滤, 这些类人思考的优

化方法，逐渐形成一种体系，不仅改变德塔作品的设计模式，同时也在改变作者的研发理念.

III II DETA Catalytic computing development 德塔催化计算发展

研发并不是每一次都是成功的，在快速傅里叶的蝶形运算优化过程中，我将离散 *DCT* 的特征进行复数编码,花了我 *1* 个月,结果失败了，想起刚开始时候我在微博里说我能将快速傅里叶计算速度加快 *200* 倍，着实打了脸，我并没有放弃，既然蝶形计算优化不成功，那么在快速左右比对小高峰过滤排序上试试？当我看到单机随机 *double* 每秒 *1200* 万数组排序速度的第 *10* 代及快速排序出来的时候，我振奋了．我的思想是对的，细化逻辑是类人思考的一种重要方式．在这里作者设计快速分词和 极快速小高峰过滤催化排序时候设计了一篇论证论文如下:

Theory on YAOGUANG's Array Split Peak Defect

Mr. Yaoguang. Luo
Liu yang deta software development limited company, hunan,china
313699483@qq.com

Outline： *In the common software development factory, engineer always did more and more interactions with data structure and math algorithms. Especially in the recursion, convolution, sort and generic loops, scientist likes to find a simple, more sufficiently and alignment way to face the project requirements with the large association. For instance me, I really got a real world problem at this domain while I use quicksort, also for other project such like DETA parser. What is the peak array split defect? How does it count the real world problems? Why need find it and how to get the nice solution? Cool, this paper will cause an implementation about our goals, ok now, keep forward to the context where I talking as below, thanks For more theory details and the source code implementations please check the bottom reference section.*

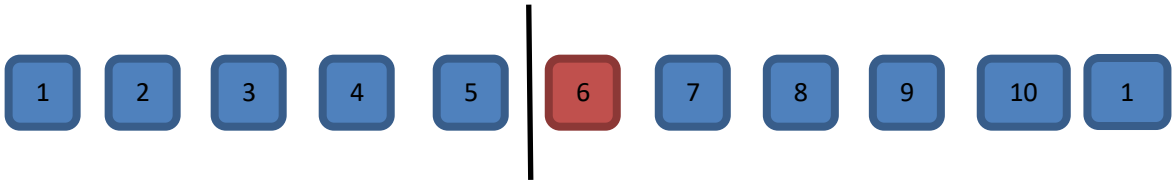
在一些主流的软件研发企业，工程师总是花费很多时间在数据结构和数学算法上,特别是在递归,演化,排序,和常规循环.科学家因此想发现一种简单高效的算法来解决一些需求，比如我，如快速排序，德塔分词，什么是非对称数列切分,这些逻辑怎么解决当前真实的社会问题？这篇文章能够解释许多问题，丰富的论据和论证以及论证的源码我都会毫不保留的提供.

Peak, Array, Split, Defect, Recursion, Convolution, Sort, Generic

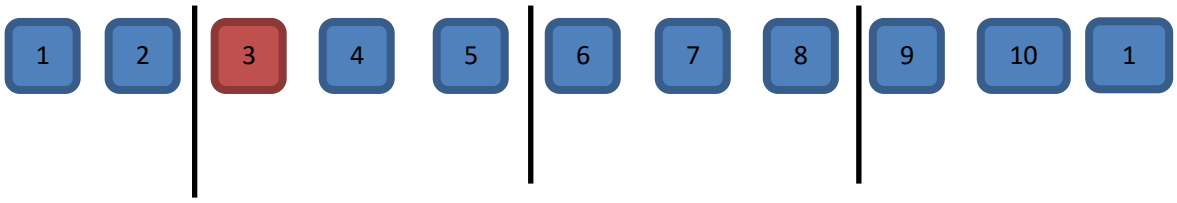
Goal one: Quicksort Yaoguang.Luo 4D

1 Details：

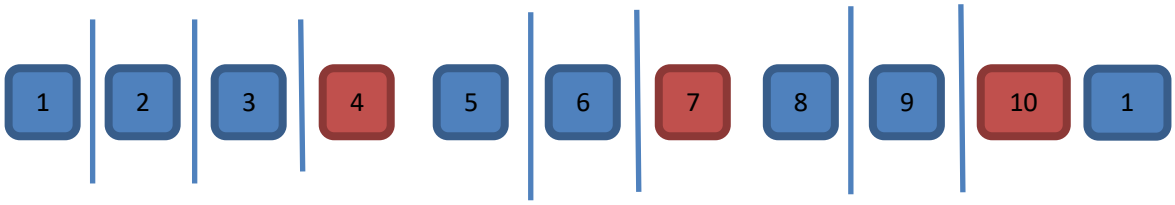
1.1 For example the array input as below where we gave 11 digits.
关于基数拆分初始，给定 *11* 个数列 队，



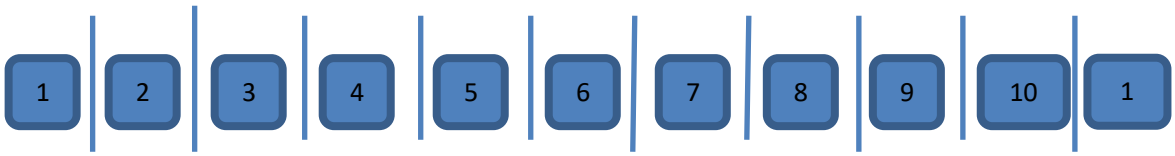
1.2 The first split, we could see the digit-6 will auto arranged to the right part.
第一次拆分如上，我们会发现 *6* 被默认在右方。



1.3 And the second split, we may see the digit-3 will be auto arranged to the right part
第二次拆分，发现 *3* 在右方。



1.4 The third split, we may see the digit-4.7.10 will be auto arranged to the right part
第三次拆分，发现 *4*，*7*，*10* 出现在右方。



2 Thinking:

After the split array showing, we could see clear that the big problem about the asymmetry defect, as I did an annotation of N, so the i of N will absolutely find a n/POW（2，i）value points, as an insufficiency asymmetry defect model, I fall in thinking... if I do any compute theory as the same with this model style, for example in the recursion or inner loops, it will autonomic separate to the 2 different process way, it necessary to do indifferent flows.

通过上面的迭代微分，可以发现一个有规律的非对称非基数缺陷问题。我用数学标识数组 *n*， *n* 为基数，则 *n* 的第 *i* 次拆分，关于这个 *n/pow*（*2*，*i*）的缺陷数，呈现不饱和对称模型，这个细节我引起了思考如果，我在做函数递归，*2* 分迭代，和矩阵微分的时候，出现了类似的基数划分，我有必要考虑通过 *2* 种算法内核形式来处理基数列和偶数的计算。

3 Problems:

So, after the above thoughts, I may get any flashes, First, the even and odd digits both are asymmetry while in the Differential loops. For this noise, I defined as (Tinoise Peak) Second，once we did a split compute under this model, it must get more unfair sets. I defined as (Tinsets defect). Third, if this model almost in the messy and timer data system, it will catch more time and asserts wastes or exceptions.

通过这个实验，我得到一些答案：*1* 不论是基数还是偶数都会增加不对称，而不是仅仅是基数的问题。这个噪声，我定义为罗瑶光拆分噪声峰。*2* 拆分的基偶划分，必然会造成内部计算算子的不平衡，我定义为罗瑶光算子缺陷。*3* 如果这个缺陷在海量数据处理中有时序性，那么这个算子缺陷会产生巨大的计算浪费。

4 Solutions:

For the god like, I find three solutions while I currently enrolled in my projects. First: computer logic acceleration, at least it can avoid the waste of the compute by using inner process optimism. -- To avoid the deep recursion. Second, reduce the compute sets. For any less memory system, we may reduce more and more memory garbages after we reduce the inner register or temp value sets. Third, we may make an optimization of the function logic where to instead the old complex functions. Those ways include the condition, algorithm, method or discrete optimization. End, we may use mathematics of double differential, deep definition, acquisition or polynomial to get the solutions.

这个细节体现在罗瑶光小高峰过滤快速算法中。*1* 增强计算逻辑,我们可以通过增加函数内部的计算性能来弥补深度迭代浪费。*2* 减少计算算子 ,我们通过减少计算临时变量和算子来避免内存垃圾增量。*3* 化简计算条件,我们可以通过函数优化，条件优化，来化简计算逻辑. *4* 重微分迭代, 我们也可以进行微分，重微分，条件微分，迭代来进行多项式展开计算。

5 True Instances

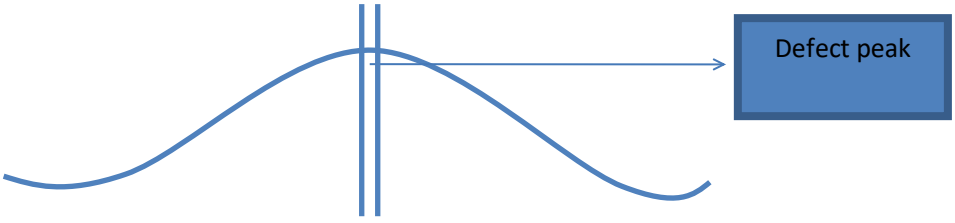
Let me show the algorithms here,

```
private int partition(int[] a, int lp, int rp) {
        int x = a[lp]< a[rp]? a[lp]: a[rp]; //reduce the compute values, reduce the recursion peak
        int lp1 = lp;
        while(lp1++ < rp){
                while(!(a[lp1++]> x|| lp1> rp)) {// reduce the condition differential check, reduce the recursion loops
                }
                while(a[rp++ ]> x){
                }
                if(-- lp1< --rp){
                        int temp= a[rp]; a[rp]= a[lp1]; a[lp1]= temp;
                }
        }
        a[lp]= a[rp]; a[rp]= x;
        return rp;
}
```

From this code: in a common quicksort way, the recursion based on the average deep split, suppose the initial array length is N{1,2,3…n} is an Odd, so the separate two arrays will cause an asymmetry defect, those timer asymmetry compute peak collection will cause more and more probability problems such like jam, lock, time waste and heap increment.

这段代码 说明在传统的快速排序中,递归总是基于平均的深度分裂，如果产生非对称的 迭代函数，我们需要讨论一些算法来过滤这些计算概率高峰，如 呼叫拥堵，时间损耗，内存堆增量等问题因素.

The odd peak binary split as below：



How to avoid those timer distinction peaks? I go more absolutely research where focus on these problems, first, differential flows. This flash is not suitable for here, May good for the DETA parser, I will show you later. Second, compute acceleration. Yep, this is a good way, for example find the big X as the code blew, it will cause the while loop ability accelerations.
怎么避免时间高峰？我有一些明确的逻辑,首先流微分，其次是计算加速，和循环优化加速.

```
int x = a[lp]< a[rp]? a[lp]: a[rp];
```

Third, De Morgan condition differential as the code below, it will cause the condition ability accelerations.
第三，离散定律变换 也能进行决策条件优化.

```
while(!(a[lp1]>x|| lp1>=rp)) {
```

At last but the least, value reduce, code optimization both are very important way of the peak avoid filter.
最后至少，算子减少，代码优化都能有效的进行计算小高峰过滤.

Goal Two: DETA parser

6 Details：

Last year I help my father to develop the study software about getting the medicine data collection for quick search. I' m going to try to build a search engine system, input format is a string, how to get a Chinese string array split?
去年,我帮我父亲研发学习软件，关于医学数据的迅捷搜索. 在设计搜索引擎的过程中,我遇到了一个问题就是格式化字符串中有效的拆分中文词汇.

6.1 convolution length indicate by marching Nero index tree as below: 2|1|2|3
如图，演化的开始按照神经森林词汇索引进行面切分,如下.

6.2 convolution POS indicate as below: n |c |n |adv |adj
切分后进行面中的词性切分如下

6.3 convolution split
词性切分后进行组合如下

7 Thinks:

While I use this way on my DETA project Chinese separations, I met so many problems, the more and more important problem is the POS frequency peak waste, my POS flow functions will spend a lot of time to do the low prior convolution split condition check first… it cost me a lot of time… after I did a collection of my projects, the results are clear.

思考了下，当我用这种方法进行德塔工程，我遇到了许多问题,特别是词性拆分算法的频率峰，很多时候高频使用的函数总在不常用的函数后面，造成了时间浪费，于是开始频率阀门排列优化.

8 Problems:

First, convolution kernel gets asymmetry problems. Second, unnecessary conditions check loops. Third, unimportant heap register values. End, values sets and conditions sets too more.

首先，演化的内核 有许多非对称的问题，其次，不必要的条件和循环检查，然后 不重要的堆注册的变量繁琐，最后静态算子,条件算子过多.

9 Solutions:

First, format the convolution kernel of the index dictionary tree, for example I use the char ASCII as the index length to reduce the match time of the convolution length indication. Also, I define the POS convolution kernel size less than five. Second, I did a condition frequency statistic, and re arrangement it, at the same time, reduced a lot of the inner sets to avoid the compute pause.

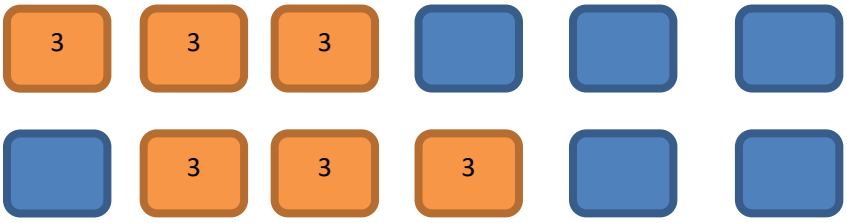
首先,格式化索引树内核,比如 *ASCII* 索引，按长度划分，同样，词性内核我定义长度不超过 *5* 个字，第二我统计了执行条件函数的频率，然后流水阀门重新排序了,高频的放在前面.依次排列. 同时,尽可能的减少内部条件算子，避免计算高峰.

10 True instance:

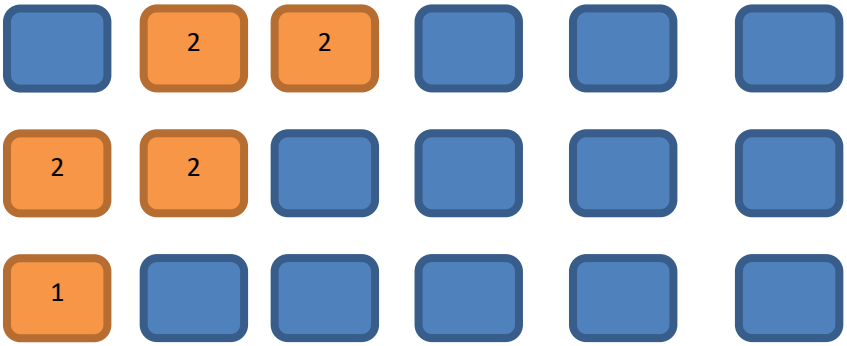
Finally I developed a convolution String array split way for marching as below: orange color are presplit sets
10.1 check 4 chars slang

最后，我设计一套内核方式，字符串词切分先 *4* 字成语谚语比较

10.2 Check 3 chars key word
不是成语然后 *3* 字比较



10.3 Check 2chars normal word
然后 *2* 字比较



最后单词拆分
10.4 In order to make a compute acceleration, I did 2 string builder array to store a pre order sets.
为了计算加速，我设计了 *2* 个数组用来存储分词坐标的前缀 词储.

10.5 in order to make a PCA POS acceleration, I did 5 chars marching array to store a post order sets.
为了进行主要词性拆分加速，我另外还设计了 *5* 个长度 后缀 字储.

It seems the Nero Index, NLP and POS for the PCA separation with convolution kernel marching by using stepwise iterative differentiation got much higher sufficiency.
计算拆分的高效性可以使用我的德塔分词的作品测试. 充分的数据论证其性能是优秀的.

III III DETA Catalytic computing application 德塔催化计算应用

为了论证细化逻辑的重要性，我开始将这种逻辑观念,融入我的养疗经和我的一切软著作品中，当我看到每秒 *1300* 万的高准确度分词，每秒 *600* 万的混合象契拼音笔画排序，每秒 *1200* 万的 *double* 数组排序，等惊人性能的作品问世，我对自己的认知开始感叹了. 我毫不保留的将这些思想和作品全部开源了. 引起人类的思维共鸣. *7* 个月前我在对 *dataswap* 作品的评论中描述
微分催化的意义 目前已经包含了 频率阀门（冯诺依曼）微分，离散逻辑（狄摩根）微分，高频函数降解，条件细化微分，执行模式微分，巨系统（钱学森）模块微分， 数学微分（牛顿，布莱尼兹）*7* 大类，是类人 *DNA* 进化算法 催化 的唯一途径。并做了详细的论证归纳如下
《微分催化计算作为类人 *DNA* 进化的唯一途径》 论证实例如下：
1 Deta 快速分词 *POS* 流水阀门微分算法 实例论证：
https://gitee.com/DetaChina/DetaParser/blob/master/wordSegment/org/tinos/engine/pos/imp/POSControllerImp.java
论证主题：微分催化计算能非常好的观测到量子态函数的执行流水逻辑，通过统计可以将高频函数逐步按冯诺依曼态提前，低频的逻辑逐步淘汰筛选。
论证结果：论证成功。减少无关代码遍历次数。大幅提高算能。

论证影响：类人 *DNA* 进化算法的进化机制主要途径。

2 罗瑶光小高峰计算过滤排序算法第 *4* 代 实例论证：

https://gitee.com/DetaChina/DataSwap/blob/dceeb0b06f726d640553964058d85b736354ac89/src/org/deta/tinos/array/LYG4DWithDoubleQuickSort4D.java

论证主题：微分催化计算能非常好的结合离散数学体系从数字逻辑层面进行微分过滤高频函数。保证平滑

论证结果：论证成功。大幅增快计算速度。

论证影响：类人 *DNA* 进化算法能有效平滑计算高峰。

3 罗瑶光欧拉森林商旅环微分 *TSP* 算法第 *2* 代 实例论证：

https://gitee.com/DetaChina/Data_Prediction/blob/master/src/org/tinos/deta/tsp/YaoguangLuoEulerRingTSP2D.java

论证主题：微分催化计算能非常好的将传统的复杂逻辑进行认知层的思维模式优化改变，一开始便从根本上改变认知过程。

论证结果：论证成功。微分催化计算在一些社会领域能改变传统认知方式。

论证影响：类人 *DNA* 进化算法能有效的选择最快的算法模块和认知模块来做特定领域的计算，提高算能。

4 罗瑶光 象契字符串条件微分排序算法第 *7* 代 实例论证：

https://gitee.com/DetaChina/DataSwap/blob/master/src/org/deta/tinos/string/LYG4DWithChineseMixStringSort7D.java

论证主题：微分催化计算能可观的将条件函数统一，减少逻辑复杂度，并能持续优化和持续专注。

论证结果：论证成功。微分催化算法能很好的将整体函数的局部模块进行 *vpcs* 逻辑拆分，形成 *DNA* 肽链的 *initon* 运算嘌呤元。

论证影响：类人 *DNA* 进化算法的自主进化机制保障。

5 Deta Socket 流可编程数据库引擎 的 *PLSQL* 语言微分编译机 实例论证：

https://gitee.com/DetaChina/Deta_PLSQL_DB/blob/master/java/org/lyg/db/plsql/imp/ExecPLSQLImp.java

论证主题：微分催化计算 多条件 执行的 可用性。

论证结果：论证成功。微分催化算法能提供 函数系统运算时 可逆向可观测运维保障。

论证影响：类人 *DNA* 进化算法的 条件微分，逻辑微分，高频阀门前置等功能的 综合集成应用实践。

上述这些论证是一年前的，目前很多作品因为作为养疗经项目的子系统研发，一直保持良好的跟进状态中.

这些岁月，我一直在思考，细化逻辑的最终表达方式是什么？我一直没有停止探索，持续的绝对专注.

IV DETA C Finding initions 德塔催化计算算子单元寻找

我一直在思考，细化逻辑的最终表达方式是什么？我一直没有停止探索，持续的绝对专注.

我一定要找到这些逻辑最终表达方式，我 *2018-2019* 年认为逻辑细化的最终表达方式一定不是 *AOPM* 和 *VPCS* 这么简单，*VPCS* 只是 *AOPM* 的一种细化层，那 *VPCS* 怎么细化呢？于是我开始整理我已有的东西，我的那些作品和软著思想. 上帝啊，我只能在我已有的基础上进行坚韧的 持续的 绝对专注. 我得赌一次.

IV I DETA C Finding initions history 德塔催化计算算子单元寻找历史

VPCS 下面是什么？函数的本质是什么？在学校,我得到了一些基础答案, *DNA* 的基元是 *ACGTU* 嘌呤 和 嘧啶，后端架构的基元是 *VPCS*，事物逻辑的基元是 *AOPM*，数据库的基元是 *IDUC* 增删改查. 函数的基元是 *IOAON* 入出与或非，我只能在我能理解的和我已有的知识结构中寻找. 怎么论证？论据怎么确定？

IV II DETA C Finding initions development 德塔催化计算算子单元寻找发展

第一个论证过程是德塔分词, *2019* 年我不断的进行分词的细化，优化，反复的提炼，发现了一写振奋人心的论点，我的分词函数进行不断的有理拆分，最后是一堆 结构比较简单的 增删改查的组合应用片段 通过 *IOAON* 的方式 来展示. 举个最有力的论证 我在分词处理名词的时候，最后函数形成了，内存拿出 *4* 字词，比较 *4* 字谚语，没有比较 *3* 字词，没有比较 *2* 字词,没有拆分成单字，这个过程 一句话概括就是按照 冯,若依曼 的时间流水形式，对内存数据 的 增，删，改，查 的组合决策过程.

我思考到这，眼前一亮， *IDUC* 不仅是数据库的操作方式，也是内存数据的操作方式，持续的绝对专注*!!!* 假设 *IDUC* 这个组合决策过程 对所有数据的操作方式 都有效，假设成功，如果编码，那这就是数据 *DNA* 的一种非常严谨的编码方式. 我找到了*!* 我开始细化我的排序算法，分词算法, *ETL*，养疗经，等作品，发现一个共同点,我的所有作品细化到我能理解的有理函数级别，都是对线性，多维，数据库，内存数据 的 增，删，改，查 的组合决策过程的小片段. 这些片段都能进行有效的编码.

IV III DETA C Finding initions application 德塔催化计算算子单元寻找应用

想到这，我的确定了一个三元的 *DNA* 对 *ETL* 神经元节点的映射编码方式 *AOPM -VPCS- IDUC 3D* 编码方式. *4*4*4* 那么每一个基元是一个 *64* 位的空间，这个空间就是我苦苦数十年要寻找的计算基元.

V DETA DNA decoding 德塔催化单元的 DNA 解码

如果 *DNA IDUC* 映射的 *AOPM -VPCS- IDUC 3D* 计算神经元 成立,那么怎么解码呢？我思考了我有什么，对养疗经！
结构 养疗经 系统,将这种思想和技术进行论证，是我目前能做的唯一方式. 还是那句话持续的不断的绝对的专注.

V I DETA DNA decoding history 德塔催化单元的 DNA 解码历史

目前人类所知道的是 *DNA* 肽团有数十亿长，双链, *24* 对染色体. *ACGTU 5* 种基元，如果 *ACGT* 能编码出人类的高等智慧逻辑.那么 *IDUC* 为单位的 类人数据 *DNA* 同样能编写出 数十万行 *AOPM VPCS* 商业的事物处理逻辑.这 *2* 种逻辑并不冲突.

V II DETA DNA decoding development 德塔催化单元的 DNA 解码发展

这两种逻辑并不冲突，难道他们本身就是一体？我得到一个振奋人心的论点，我得找到论据,于是我将最近这 *20* 年的学习和工作方式，我的本身思维方式和 我的软件的执行逻辑方式进行筛选，希望能找到否定的理论来推翻它，可惜我没有找到. 于是我跟进了一步，重新审阅了我的软著作品，优化它们，发现一旦优化到有理函数到无理函数的边缘，都是对线性，多维，数据库，内存数据 的 增，删，改，查 的组合决策过程的小片段. 这些片段都能进行有效的编码 *AOPM -VPCS- IDUC* 似乎能解释一切我要的答案.

V III DETA DNA decoding application 德塔催化单元的 DNA 解码应用

为了推翻我的这个论点，我开始到处寻找论据来攻击这个论点，首先我找了生命永生的话题，按照 *AOPM -VPCS- IDUC* 来理解 便是 *IDMC* 为 *ture*，既然能完美的解释，于是我又找了新冠的感染话题，便是 *DUOP* 为 *ture*，令人振奋人心的结论！我苦寻数十年的问题全部能从 *AOPM -VPCS- IDUC* 中找到答案，于是我开始了映射编码如下:

I 增加	D 删除	U 改变	C 查找
V 视觉	P 注册	C 控制	S 静态
A 分析	O 操作	P 处理	M 管理

永生的肽团可以筛选出

I 增加	***D 删除***	U 改变	C 查找
V 视觉	P 注册	***C 控制***	S 静态
A 分析	O 操作	P 处理	***M 管理***

人类特征新陈代谢相关的肽团可以筛选出

I 增加	***D 删除***	U 改变	C 查找
V 视觉	P 注册	C 控制	***S 静态***
A 分析	O 操作	P 处理	***M 管理***

新冠相关的肽团可以筛选出

I 增加	***D 删除***	***U 改变***	C 查找
V 视觉	***P 注册***	C 控制	S 静态
A 分析	***O 操作***	P 处理	M 管理

人类过敏反应 相关的肽团可以筛选出

I 增加	D 删除	U 改变	***C 查找***
V 视觉	P 注册	C 控制	S 静态
A 分析	O 操作	***P 处理***	M 管理

是的我无法推翻这个编码规范...

我得到一个清晰的 DNA 定理: DNA 的本质是一种智慧体 对 数据 增删改查 的 四个 元操作 的 组合方式 编码索引.

2020 年 10 月 4 日

VI IDUC DNA and Its Applications 类人 DNA 进化过程 在真实环境中的应用 分析 疑问.

这些都是后话. 应用面太广泛了，首先，我的 *ETL* 开始往 *3* 维方向拓展更好的为医学服务. 其次，病毒免疫学 和 永生病毒

探索, 永不停歇. 为什么采用 *ETL* 来作为拓展点, 灵感来自于我 *2013* 年 *10* 月 *17* 日一篇 *osgi* 的感想论文.如下

论证作品： 人工智能的达尔文进化思想（*The Darwin's Theory of The Artificial Intelligence*）

最新的知识工程结构中，传统的专家系统占据着主导的地位，可是世界的需求体系处在一个多变的运行环境，所以数据持久化理论是一个为之奋斗的目标。人工智能软件也一样，逃避不了自然的更新所带来的种种弊端。人工智能何去何从，自然会规划它，正如达尔文的生物进化论一样，新的智能体系标准都是被需求自然选择出来，这就是我要表达的中心思想。过去 *50* 年里，一些经典的软件逃不过需求的抉择，最终枯黄暗淡，当然一些企业将产品拼命的重写升级，因为核心开发者的年龄老化，新的改造者无法掌握原始开发思想和理论，最后产品的品质遭受巨大的冲击，损失惨重，一种新的软件开发理论需要被人所证实，这也就是我的思想。软件也一样，需要有自我的人工选择的进化体系。

通过最近的 *UNICORN AI* 软件的构造，设计和编码测试中，我发现了许多因空想而创造的计算机理论在实际的编程分析中有巨大的差异，我用的是 *JAVA* 为主的语言，我就发现 *JAVA* 的继承没有达到具有进化思想的语言标准，但是 *JAVA* 在这个初期的进化标准测试中其方法论远远胜出 *C/C++*,我用 *C* 风格写 *JAVA* 程序并没有给我的实际编程带来种种麻烦，但是 *JAVA* 仍然需要改进，比如你抽象了一个父类，而你的子类的变量函数还是需要在*"OBJECT* 父类=（子类）父类*"* 这样的写法中的才能做出子类运算。如果孙类又继承子类，怎么让 *OBJECT* 得到孙类？（我的用的是 *OBJECT* 子类继承父类，然后 *OBJECT* 子类=（孙类）子类 。这样孙类得到了运算），可是这就是一个动态内存结构分配的大问题！设计的相当繁琐。*JAVA* 还停留在初级语言进化级别，没有具备高级的进化思想。其次， 子类如果有多个孙类，也只有子类可以运算，父类就被无法作出相应的运算。这也是一个诟病，难道再加上 *OBJECT* 子类=（孙类）子类 ，*OBJECT* 父类=（子类）父类 来实现？这就更加繁琐了。

通过上面的描述，我有自己的看法，可是我还是选择了 *JAVA*, 即使繁琐，但是没有任何错误，因为用底层语言来实现就会更加繁琐。陷阱更多。人工智能选择了 *JAVA* 是一个自然的抉择。*JAVA* 和 *C#*都是高层语言，可是 *JAVA* 的个性就是天生对数据来处理的，因为 *JAVA* 早期是一个 *WEB* 语言，*WEB* 处理数据信息有独特的优势，这是 *JAVA* 进化为数据分析语言的一个真实的例子。*C#*在这个问题里一直在改进自己，类似 *JAVA* 一样，甚至和 *JAVA* 一样，可是没有一个体系来评估它。早期应用 *JAVA* 的 *WEB* 数据工程师也不会转移到 *C#*.所以 *C#*的最大优势还是仅仅在 *WINDOWS* 上的控件应用。

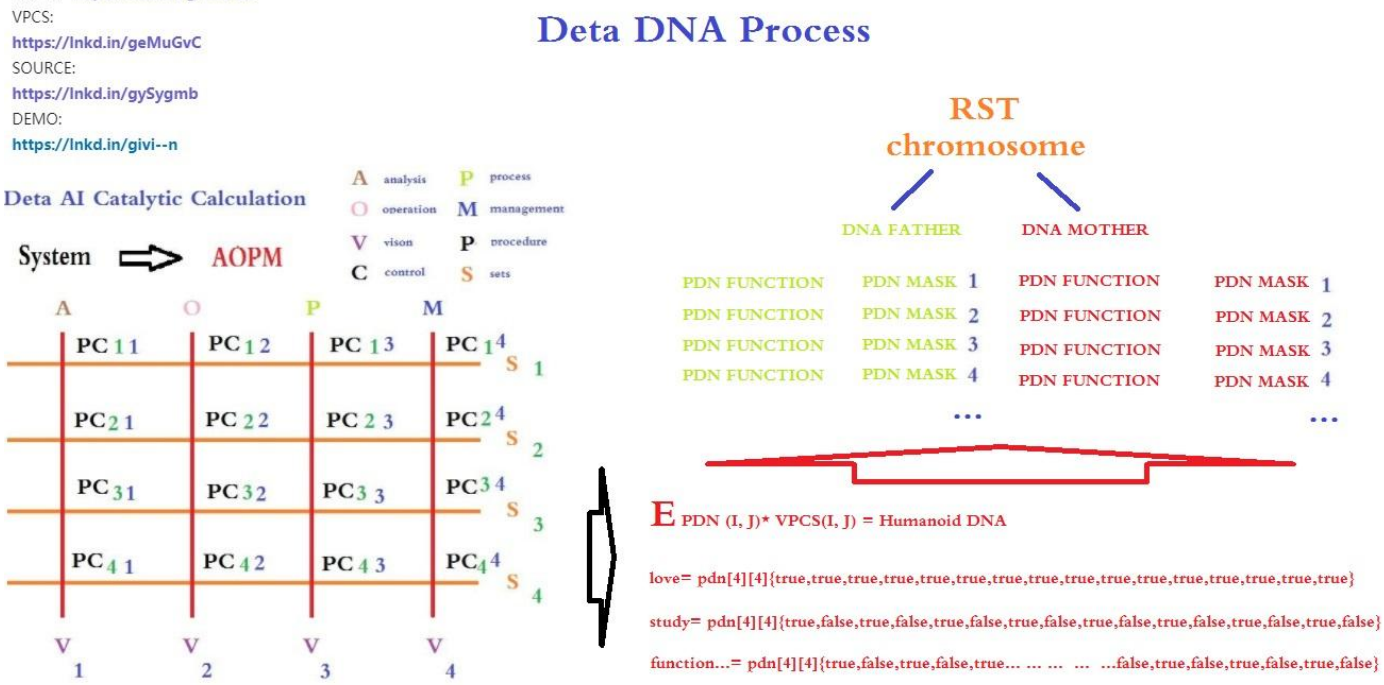
通过这段的描述，仅仅证明任何一种语言的最大优势也仅仅体现在它诞生之初的创造理论和思想。所以 *JAVA* 和 *C#*根本就没有什么可比性。因为他们最原始的创造理论，体系和思想结构就不一样。如果真的 *JAVA* 和 *C#*不倒，最后，通过进化的思想预测，*JAVA* 最后走图形，大数据分析，*WEB*,方向， 而 *C#* 应该走界面，控件，*WINDOWS* 设备集成方向。人工智能软件的进化主要分为父类的更新，子类的变异和继承。现在的许多人工智能软件因为需求关系的制约，导致创造思想的缺乏，父类被写死了，无法得到应有的适应扩展.

JAVA 处理子类函数是比较完美的，用过 *JAVA* 开发大型项目的人都相当有经验处理接口和继承。可是 *JAVA* 有没有变异的特性呢？可以说无，比如我举个例子，当父类 *PUBLIC* 属性 *1=0*;，子类就无法在 *PUBLIC* 属性 *1=1* 了，这就是一个变异失效的问题。*JAVA* 很灵活，但是不够脚本语言灵活。其次我要说的是 *JAVA* 的变异是带引号的变异，其特点就是子类修改父类函数，*JAVA* 的子类是可以修改父类的同名函数处理过程的。不过你要让子类和父类的函数名一样，这是一个 *JAVA* 默认的机制，先执行父类同名，再执行子类同名。然后返回到父类，然后返回的过程。所以同名函数可以在子类里得到修改，保证了参数变异。这样，软件在实际的编写过程中也非常的灵活和独到。最后通过上述的语言进化思想，程序进化思想的表述，我有一个很深的体会。每一种语言要根深蒂固，需要有它的需求，它的功能在需求中要有选择的得到进化。不然，这就是语言被淘汰的最大原因，我不喜欢看到当今世界上各种语言层出不穷，这就是许多语言没有得到进化，体现不了需求的最大诟病。 其次，语言需要扩展，高级语言的 *API* 类库和一些架构体系的出现是一个很好的扩展证明。最后是变异，类似脚本语言，灵活，方便。

那么软件呢？软件也一样，选择一门适应自己需求的语言来设计尤为重要。 其次，软件的架构要有松耦度，类似于 *OSGI,FELIX* 那样,进行组件持久化，*KNIME* 的 *OSGI* 思想和 *LIFERAY* 的 *OSGI* 思想是一致的，虽然 *API* 设计风格不一样，但是效果都很笃厚。 生物需要有达尔文思想，人工智能同样也存在，这是需求持久化的基础。这也是我研发 *UNICORN AI* 平台的基本条件。

现在我有足够信心持续专注在让我的带进化系统的 *DNA* 编码所映射的 *ETL* 重用 完美的保障 需求持久化 论点. 关于怎么用 *ETL* 来映射编码,我会再次回到前年,分析当时这张图片的设计思路如下*:*

AOPM: https://lnkd.in/gmfRFGS
VPCS:
https://lnkd.in/geMuGvC
SOURCE:
https://lnkd.in/gySygmb
DEMO:
https://lnkd.in/givi--n



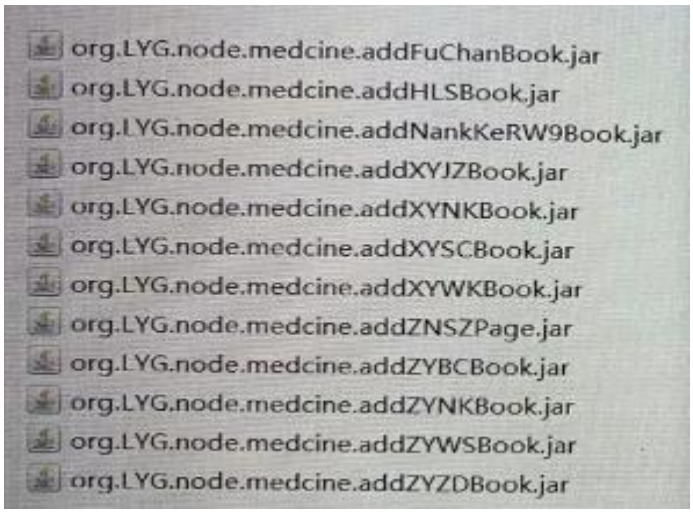
是的我已经有想法了!! 令人兴奋, *ETL* 节点 三维分类. 这个词汇.

***VII IDUC VPCS AOPM 3D Nero Cell and Its Applications* 德塔 *DNA* 编码规范怎么在应用中论证**

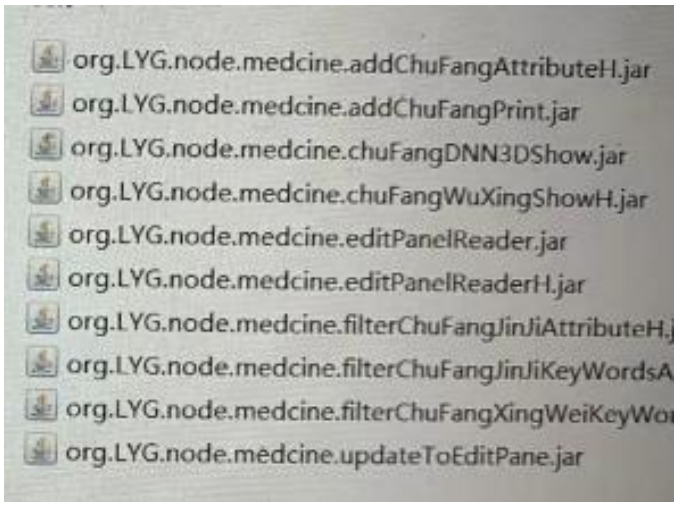
这些也是后话, *ETL* 开始往 *3* 维方向拓展, 首先我要做的是基于德塔 *IDUC DNA* 映射 的 神经元的 *3D* 功能区设计. 这是我所能理解的真正的类人自主思考方式.

具体 养疗经 应用实例

养疗经软件 的 书籍节点导入我会分类到 操作 静态 查找 增加 的节点分类中.



养疗经软件 的 医学专科节点导入我会分类到 操作 注册 静态 增加 的节点分类中.
养疗经软件 的 处方分析节点导入我会分类到 分析 视觉 控制 改变 的节点分类中.



这就是 DNA 编码手册的人类历上第一次 索引应用思路.
我会把 *Org.lyg.node.medcine.addchufangattributeH.jar* 修改成 *Org.node.a.v.c.u.medcine.addchufangattributeH.jar*
这个 *a.v.c.u* 会形成一个分析 视觉 控制 改变的 *dna* 映射体系编码. 方便之后的进化优化测试. 再之后这些 *ETL* 索引映射集合 我会系统编码成 养疗经的*DNA* 索引链. 这篇论文给我的思路都是创世的思路. 感谢一切. 关于编码的整体性我可以先设计成 *AOPM VPCS IDUC initons 64* 位单链, 如

AVI AVD AVU AVC API APD APU APC ACI ACU ACD ACC ASI ASD ASU ASC
OVI OVD OVU OVC OPI OPD OPU OPC OCI OCD OCU OCC OSI OSD OSU OSC
PVI PVD PVU PVC PPI PPD PPU PPC PCI PCD PCU PCC PSI PSD PSU PSC
MVI MVD MVU MVC MPI MPD MPU MPC MCI MCD MCU MCC MSI MSD MSU MSC

这个索引方式, 即使不是最终的人类生物 有机 DNA 对应索引, 但是它已经成为第一次人类人工设计代表可进化编码的 DNA 的映射执行方式.

Not The End

结论:

这篇论文，我花了20年的基础学习，7年的工作实践，2年的开源实现，18个相应数据子工程，6个数据领域软著，2个大数据作品，7篇人工智能论文，逐步论证一些本质，如

DNA 定理: DNA 的本质是一种智慧体 对 数据 增删改查 的 四个 元操作 的 组合方式 编码索引.

另外:

神经元计算执行方式: DNA 编码索引的 映射的 具体功能计算的 神经元时序计算链

适应环境的本质: 就是 DNA 编码索引 映射相关的神经元 实现更好的增删改查.

类人进化子代的本质: 对数据高效处理的神经元计算方式 所映射保留的 DNA 编码索引链中 相同编码逻辑部分的 进行优化杂交 所产生的子代. .

*话题似乎永远不会结束，不妨大胆的提出新论点 持续不断的坚韧的专注.论证. 而我依然乐在其中^_^*

VIII Refer:

德塔分词 https://github.com/yaoguangluo/Deta_Parser

德塔 ETL https://github.com/yaoguangluo/ETL_Unicorn

德塔 Socket 流数据库 https://github.com/yaoguangluo/Deta_DataBase

德塔极快速排序 https://github.com/yaoguangluo/sort

软件工程 https://baike.sogou.com/kexue/d10131.htm?ch=fromsearch

MVC https://baike.sogou.com/v25227.htm?fromTitle=MVC

MVP https://baike.sogou.com/v70887934.htm?fromTitle=MVP%E6%A8%A1%E5%BC%8F

ORACLE https://baike.sogou.com/v110535.htm?fromTitle=oracle

VPCS https://github.com/yaoguangluo/Deta_Resource/blob/master/VPCS-Method_V1.1.doc

AOPM https://github.com/yaoguangluo/Deta_Resource/blob/master/AOPM%20System%20On%20VPCS.doc

养疗经 *http://tinos.qicp.vip/Deta_HuaRuiJi.html*

华瑞集 *http://tinos.qicp.vip/Deta_HuaRuiJi.html*

SONAR https://www.sonarlint.org/

DNA

https://baike.baidu.com/item/%E8%84%B1%E6%B0%A7%E6%A0%B8%E7%B3%96%E6%A0%B8%E9%85%B8/78250?fromtitle=dna&fromid=98123&fr=aladdin

达 尔 文

https://baike.baidu.com/item/%E6%9F%A5%E5%B0%94%E6%96%AF%C2%B7%E7%BD%97%E4%BC%AF%E7%89%B9%C2%B7%E8%BE%BE%E5%B0%94%E6%96%87/82699?fromtitle=%E8%BE%BE%E5%B0%94%E6%96%87&fromid=23890

神经元 *https://baike.baidu.com/item/%E7%A5%9E%E7%BB%8F%E5%85%83/674777?fr=aladdin*

人工智能 https://baike.baidu.com/item/%E4%BA%BA%E5%B7%A5%E6%99%BA%E8%83%BD/9180

编码解码 https://baike.baidu.com/item/%E7%BC%96%E7%A0%81%E8%A7%A3%E7%A0%81

并行计算 https://baike.baidu.com/item/%E5%B9%B6%E8%A1%8C%E8%AE%A1%E7%AE%97

类人仿生 https://baike.baidu.com/item/%E4%BB%BF%E7%94%9F%E4%BA%BA/5142593?fr=aladdin

数据挖掘 https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E6%8C%96%E6%8E%98/216477

UNICORN AI https://github.com/yaoguangluo/ETL_Unicorn

KNIME

https://www.baidu.com/link?url=g_8i8yfDrvNMqYfN6A9z_XoIc49s8yqzHgpYn-JCBIi&wd=&eqid=88da123b0000be2800000006 5f7eb5df

OSGI https://www.oschina.net/p/osgi?hmsr=aladdin1e1

DML

https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E6%93%8D%E7%BA%B5%E8%AF%AD%E8%A8%80/10826467?fromtitle=DML&fromid=10035808&fr=aladdin

冯,若依曼 https://baike.baidu.com/item/%E7%BA%A6%E7%BF%B0%C2%B7%E5%86%AF%C2%B7%E8%AF%BA%E4%BE%9D
蝶形计算 https://baike.baidu.com/item/%E8%9D%B6%E5%BD%A2%E8%BF%90%E7%AE%97/4756906
多线程 *https://baike.baidu.com/item/%E5%A4%9A%E7%BA%BF%E7%A8%8B*
蓝牙 https://baike.baidu.com/item/%E8%93%9D%E7%89%99
肽团/链 https://baike.baidu.com/item/%E8%82%BD%E9%93%BE/8625112?fr=aladdin
SDLC: "", https://en.wikipedia.org/wiki/Systems_development_life_cycle
VPCS: Yaoguang. Luo, https://github.com/yaoguangluo/VPCS_Theroy/blob/master/VPCS-Method_V1.1.pdf
OSS Book: Eric Steven Raymond, 《The Cathedral and the Bazaar》
DETA parser source from GITEE: https://gitee.com/DetaChina/DetaParser
DETA parser source from GITHUB: https://github.com/yaoguangluo/Deta_Parser
DETA parser split method record: https://github.com/yaoguangluo/Deta_Parser/issues/21
DETA parser official demo: http://tinos.qicp.vip/data.html
DETA parser integrated products demo from GITHUB: https://github.com/yaoguangluo/Deta_Medicine
DETA parser integrated products demo from GITEE: https://gitee.com/DetaChina/Deta_Medicine
DETA parser integrated products demo download link: http://tinos.qicp.vip/download/HuaRuiJiTm1.0.3.zip
Quicksort YAOGUANG.LUO 4D source from GITHUB :
https://github.com/yaoguangluo/Data_Processor/blob/master/DP/sortProcessor/Quick_Luoyaoguang_4D.java
Quicksort YAOGUANG.LUO 4D source from GITEE:
https://gitee.com/DetaChina/Deta_Data_Processor_Pub/blob/master/DP/sortProcessor/Quick_Luoyaoguang_4D.java
Reflection on YAOGUANG's Peak Array Split Defect1.0
https://github.com/yaoguangluo/Deta_Resource/blob/master/Reflection%20on%20Yaoguang's%20Peak%20Array%20Split%20Defect1.0.pdf
https://github.com/yaoguangluo/DETA_DataBase , this project is building based on VPCS 1.0 theory. From this project, sleeper is more likes a socket thread, Hall is more like a sleepers pool, the vision more like a http call, and dream are more like a database system management. For any question please check the reference links as bellows:
https://github.com/yaoguangluo/NeroParser , this project demonstrated the Chinese and English words parser for the NLP AI domain, it mostly uses VPC to build the algorithms engine and kept 27, 000, 000 mixed words parsers per each second.
https://github.com/yaoguangluo/Deta_VPCS_Frontend, this project is a high sufficiency and tiny HTTP socket web, it only contains java script and java source file. Boot takes 9ms on the VPCS engine.
https://github.com/yaoguangluo/DETA_CACHE, this project is a rest calls cache by using VPCS TCP method. It seems very tiny and fast.
https://github.com/yaoguangluo/DETA_BackEnd, this projects is a backend rest service by using VPCS.
https://github.com/yaoguangluo/ETL_Unicorn,this, project is a data mining ETL software by using VPC, similar with the KNIME. Author makes high recommendation about this portrait.

Thanks GIT Hub, Eclipse.org, Microsoft, Apache and Java. Google, Angular, JQUERY, Bootstrap, JSON, Oracle.
Thanks from the bottom of my deep heart.
Thanks to my family, my colleagues, and my customers. Common!