Contents

[NFS: Network File System 2](#_Toc152188084)

[LDAP,SSSD 3](#_Toc152188085)

[EFK: Elasticsearch + Fluentd + Kibana 3](#_Toc152188086)

[ElasticSearch: 日志索引 3](#_Toc152188087)

[Fluentd: Unified Logging Layer, 日志采集 9](#_Toc152188088)

[Kibana: 日志展示和分析 9](#_Toc152188089)

[K8S部署EFK 10](#_Toc152188090)

[VDI (virtual desktop infrastructure): ETX, FastX 16](#_Toc152188091)

[Tools 19](#_Toc152188092)

[Fortify 19](#_Toc152188093)

[BlackDuck 19](#_Toc152188094)

[Nexus 19](#_Toc152188095)

[Drools 19](#_Toc152188096)

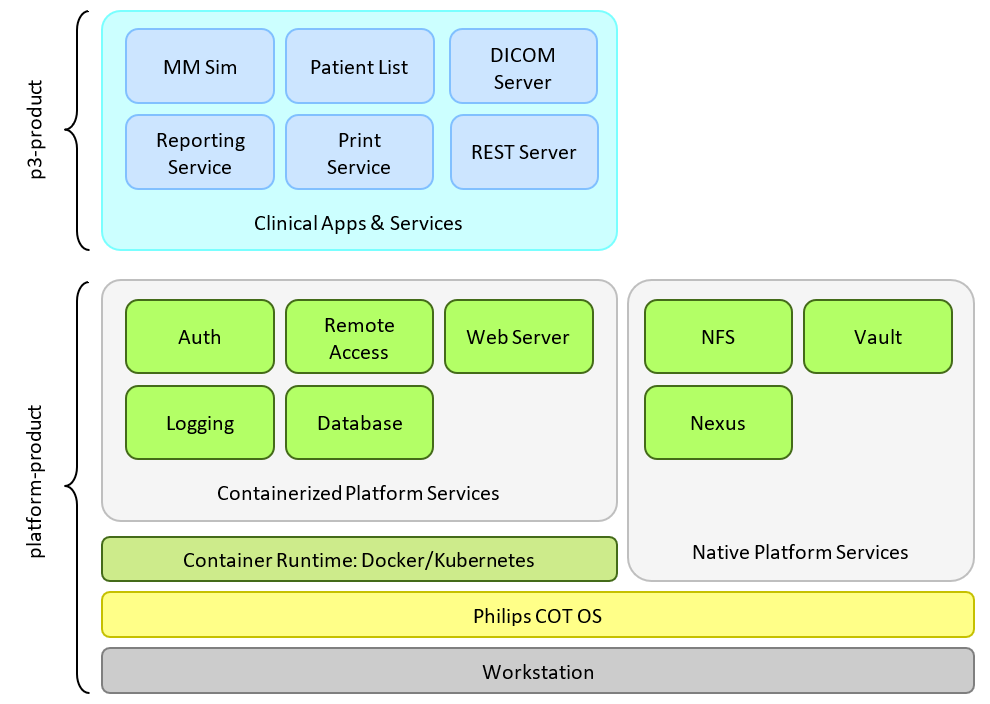


Figure. Design Overview of MMSIM

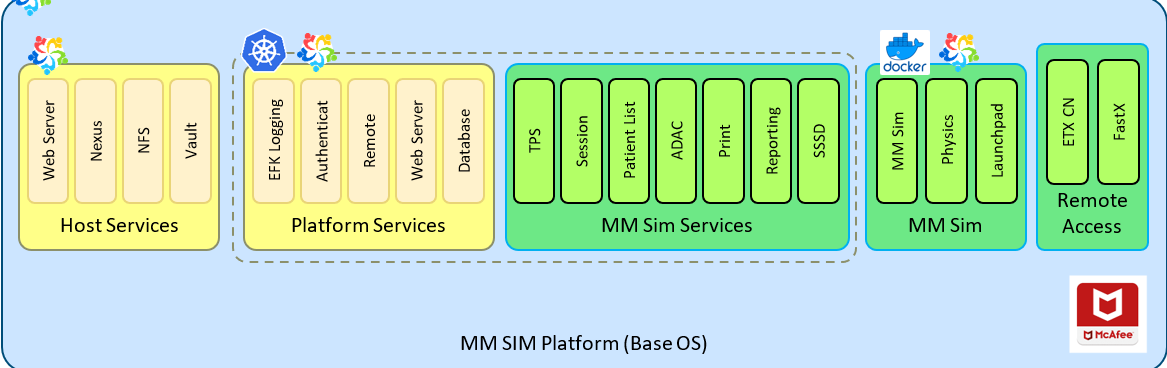


Figure. Platform service

# NFS: Network File System

文件在不同主机之间的传输大多是使用FTP服务器软件来进行传送，不过文件在 Server 与 Client 端都会存在。能否在 Client 端直接操作Server端文件？

* **NFS（Network File System网络文件系统）：**仅允许网络中的Unix-like机器之间共享文件资源。将NFS主机分享的目录，挂载到本地客户端当中，本地NFS的客户端应用可以透明地读写位于远端NFS服务器上的文件，在客户端端看起来，就像访问本地文件一样。
* 网上邻居：仅Windows 系统的机器之间共享文件资源。
* **SAMBA**：为局域网内的不同计算机之间提供文件及打印机等资源的共享服务

# LDAP,SSSD

**LDAP**, the Lightweight Directory Access Protocol, is standards-based mechanism for interacting with directory servers

LDAP的一个常用用途是单点登录，用户可以在多个服务中使用同一个密码，通常用于公司内部网站的登录中（这样他们可以在公司计算机上登录一次，便可以自动在公司内部网上登录）。

LDAP目录与普通数据库的主要不同之处在于数据的组织方式，它是一种有层次的、树形结构.

SSSD: System Security Services Daemon

本地客户通过SSSD连接认证服务器（如LDAP）,取得认证与授权信息

# EFK: Elasticsearch + Fluentd + Kibana

日志是线上业务运行的重要指标, 通过日志可以提供错误堆栈、调用链路记录、服务器运行指标、用户画像（例如浏览器版本、用户停留时间、页面性能指标）等信息.

## ElasticSearch: 日志索引

**Introduction**

**大部分数据库**能够通过时间戳或者精确匹配做过滤，但是**无法全文搜索，处理同义词和根据相关性给文档打分，无法根据同一份数据生成分析和聚合**的结果，**无法做到对数据的实时处理**。

Elasticsearch能够执行全文搜索（非结构化数据）、结构化搜索（结构化数据）和分析，是一个分布式、RESTful 风格的搜索和数据分析引擎。通常用于索引和搜索大量日志数据。

**install and run**

$wget <https://download.elastic.co/elasticsearch/release/org/elasticsearch/distribution/tar/elasticsearch/2.3.5/elasticsearch-2.3.5.tar.gz>

$tar zxvf elastic…gz

$cd elasti…gz

$tree 查看文件目录结构

./bin/plugin -i elasticsearch/marvel/latest //install marvel

若需要代理:

./bin/plugin -Dhttps.proxyHost=https://165.225.96.34/ -Dhttps.proxyPort=10015 \

install elasticsearch/marvel/latest

$./bin/plugin install -h

//On Linux and Mac

$./bin/plugin -DproxyHost=host\_name -DproxyPort=port\_number \

--install mobz/elasticsearch-head

//On Windows

$ set JAVA\_OPTS="-DproxyHost=host\_name -DproxyPort=port\_number"

$.bin/plugin --install mobz/elasticsearch-head

./bin/elasticsearch -d //run elasticsearch

curl 'http://localhost:9200/?pretty' //client

curl -XPOST 'http://localhost:9200/\_shutdown' //shut down elasticsearch

**与Elasticsearch交互**

Java API

节点客户端(node client)：节点客户端以无数据节点(none data node)身份加入集群，换言之，它自己不存储任何数据，但是它知道数据在集群中的具体位置，并且能够直接转发请求到对应的节点上。

传输客户端(Transport client)：这个更轻量的传输客户端能够发送请求到远程集群。它自己不加入集群，只是简单转发请求给集群中的节点。

两个Java客户端都通过9300端口与集群交互，使用Elasticsearch传输协议(Elasticsearch Transport Protocol)。集群中的节点之间也通过9300端口进行通信。

基于HTTP协议，以JSON为数据交互格式的RESTful API

其他所有程序语言都可以使用RESTful API，通过9200端口的与Elasticsearch进行通信

curl -X<VERB> '<PROTOCOL>://<HOST>:<PORT>/<PATH>?<QUERY\_STRING>' -d '<BODY>'

VERB HTTP方法：GET, POST, PUT, HEAD, DELETE

PROTOCOL http或者https协议（只有在Elasticsearch前面有https代理的时候可用）

HOST Elasticsearch集群中的任何一个节点的主机名，如果是在本地的节点，那么就叫localhost

PORT Elasticsearch HTTP服务所在的端口，默认为9200

PATH API路径（例如\_count将返回集群中文档的数量），PATH可以包含多个组件，例如\_cluster/stats或者\_nodes/stats/jvm

QUERY\_STRING 一些可选的查询请求参数，例如?pretty参数将使请求返回更加美观易读的JSON数据

BODY 一个JSON格式的请求主体（如果请求需要的话）

Elasticsearch是面向文档(document oriented)的，这意味着它可以存储整个对象或文档(document)。然而它不仅仅是存储，还会索引(index)每个文档的内容使之可以被搜索。在Elasticsearch中，你可以对文档（而非成行成列的数据）进行索引、搜索、排序、过滤

**Relational DB -> Databases -> Tables -> Rows -> Columns**

**Elasticsearch -> Indices -> Types -> Documents -> Fields**

curl -i -XPUT 'http://localhost:9200/megacorp/employee/1?pretty' -d '{}' 插入文档

curl -i -XGET 'http://localhost:9200/megacorp/employee/1?pretty' 获取文档

搜索全部:默认情况下搜索会返回前10个结果

curl -i -XGET 'http://localhost:9200/megacorp/employee/\_search?pretty'

//查询字符串(query string)搜索

curl -i -XGET 'http://localhost:9200/megacorp/employee/\_search?pretty&q=last\_name:Smith'

//使用DSL语句查询

不再使用查询字符串(query string)做为参数，而是使用请求体代替

curl -i -XGET 'http://localhost:9200/megacorp/employee/\_search?pretty' -d

'{

"query": {

"match": {

"last\_name": "Smith"

}

}

}'

找到姓氏为“Smith”的员工，and年龄大于30岁的员工

GET /megacorp/employee/\_search

{

"query" : {

"filtered" : {

"filter" : {

"range" : {

"age" : { "gt" : 30 } <1>

}

},

"query" : {

"match" : {

"last\_name" : "smith" <2>

}

}

}

}

}

// 全文搜索

默认情况下，Elasticsearch根据结果相关性评分来对结果集进行排序，所谓的「结果相关性评分」就是文档与查询条件的匹配程度.相关性(relevance)的概念在Elasticsearch中非常重要，而这个概念在传统关系型数据库中是不可想象的，因为传统数据库对记录的查询只有匹配或者不匹配。

GET /megacorp/employee/\_search

{

"query" : {

"match" : { //匹配rock or climbing

"about" : "rock climbing"

}

}

}

GET /megacorp/employee/\_search

{

"query" : {

"match\_phrase" : { //匹配rock climbing

"about" : "rock climbing"

}

}

}

//从每个搜索结果中高亮(highlight)匹配到的关键字，这样用户可以知道为什么这些文档和查询相匹配

GET /megacorp/employee/\_search

{

"query" : {

"match\_phrase" : {

"about" : "rock climbing"

}

},

"highlight": {

"fields" : {

"about" : {}

}

}

}

// 分析

Elasticsearch有一个功能叫做聚合(aggregations)，它允许你在数据上生成复杂的分析统计

聚合也允许分级汇总

统计每种兴趣下职员的平均年龄

GET /megacorp/employee/\_search

{

"aggs" : {

"all\_interests" : {

"terms" : { "field" : "interests" },

"aggs" : {

"avg\_age" : {

"avg" : { "field" : "age" }

}

}

}

}

}

**集群内部工作方式**

集群和节点

节点(node)是一个运行着的Elasticsearch实例。集群(cluster)是一组具有相同cluster.name的节点集合，他们协同工作，共享数据并提供故障转移和扩展功能

你最好找一个合适的名字来替代cluster.name的默认值，比如你自己的名字，这样可以防止一个新启动的节点加入到相同网络中的另一个同名的集群中。

Elasticsearch用于构建高可用和可扩展的系统。扩展的方式可以是购买更好的服务器(纵向扩展(vertical scale or scaling up))或者购买更多的服务器（横向扩展(horizontal scale or scaling out)）。

对于大多数数据库而言，横向扩展意味着你的程序将做非常大的改动才能利用这些新添加的设备。对比来说，Elasticsearch天生就是分布式的：它知道如何管理节点来提供高扩展和高可用。这意味着你的程序不需要关心这些。

做为用户，我们能够与集群中的任何节点通信，包括主节点。每一个节点都知道文档存在于哪个节点上，它们可以转发请求到相应的节点上。我们访问的节点负责收集各节点返回的数据，最后一起返回给客户端。这一切都由Elasticsearch处理。

curl -XGET 'http://localhost:9200/\_cluster/health' //集群健康

集群健康有三种状态：green、yellow或red。

green 所有主要分片和复制分片都可用

yellow 所有主要分片可用，但不是所有复制分片都可用

red 不是所有的主要分片都可用

为了将数据添加到Elasticsearch，我们需要索引(index)——一个存储关联数据的地方。实际上，索引只是一个用来指向一个或多个分片(shards)的“逻辑命名空间(logical namespace)”.

分片就是一个Lucene实例，并且它本身就是一个完整的搜索引擎。我们的文档存储在分片中，并且在分片中被索引，但是我们的应用程序不会直接与它们通信，取而代之的是，直接与索引通信。

你索引中的每个文档属于一个单独的主分片，所以主分片的数量决定了索引最多能存储多少数据。

复制分片只是主分片的一个副本，它可以防止硬件故障导致的数据丢失，同时可以提供读请求，比如搜索或者从别的shard取回文档。

当索引创建完成的时候，主分片的数量就固定了，但是复制分片的数量可以随时调整。

默认情况下，一个索引被分配5个主分片

调整：分配3个主分片和一个复制分片（每个主分片都有一个复制分片）

PUT /blogs

{

"settings" : {

"number\_of\_shards" : 3,

"number\_of\_replicas" : 1

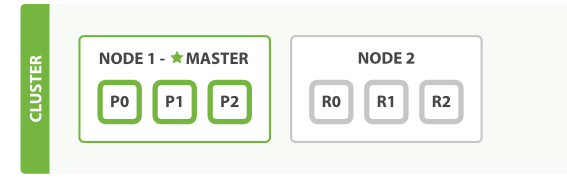
}

}

集群的健康状态yellow表示所有的主分片(primary shards)启动并且正常运行了——集群已经可以正常处理任何请求——但是复制分片(replica shards)还没有全部可用。事实上所有的三个复制分片现在都是unassigned状态——它们还未被分配给节点。在同一个节点上保存相同的数据副本是没有必要的，如果这个节点故障了，那所有的数据副本也会丢失。

增加故障转移

只要第二个节点与第一个节点有相同的cluster.name（请看./config/elasticsearch.yml文件），它就能自动发现并加入第一个节点所在的集群。如果没有，检查日志找出哪里出了问题。这可能是网络广播被禁用，或者防火墙阻止了节点通信。

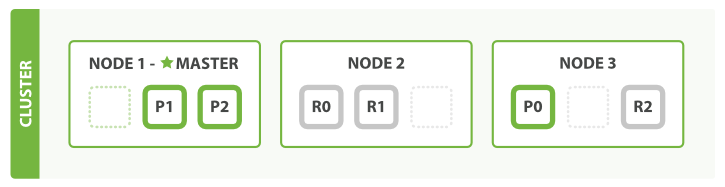


文档的索引将首先被存储在主分片中，然后并发复制到对应的复制节点上。这可以确保我们的数据在主节点和复制节点上都可以被检索。

curl -XGET 'http://localhost:9200/\_cluster/health' //集群健康

横向扩展

启动第三个节点，我们的集群会重新组织自己



Node3包含了分别来自Node 1和Node 2的一个分片，这样每个节点就有两个分片，和之前相比少了一个，这意味着每个节点上的分片将获得更多的硬件资源（CPU、RAM、I/O）。

分片本身就是一个完整的搜索引擎，它可以使用单一节点的所有资源。我们拥有6个分片（3个主分片和三个复制分片），最多可以扩展到6个节点，每个节点上有一个分片，每个分片可以100%使用这个节点的资源。

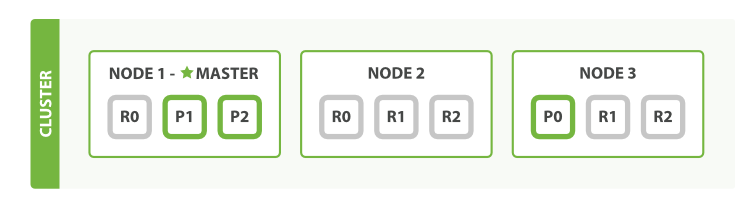
复制分片的数量可以在运行中的集群中动态地变更，这允许我们可以根据需求扩大或者缩小规模。让我们把复制分片的数量从原来的1增加到2：

PUT /blogs/\_settings

{

"number\_of\_replicas" : 2

}



## Fluentd: Unified Logging Layer, 日志采集

Fluentd 是一个日志收集系统，通过获取容器日志文件、过滤和转换日志数据，然后将数据传递到 Elasticsearch 集群，在该集群中对其进行索引和存储。

一条日志消息在 Fluentd 中被看成一个 Event 事件，Fluentd 的事件主要由下面三部分组成：

* 标签 tag：用于描述事件来源，可用于后面的事件路由
* 时间 time：事件发生的时间，时间格式为 Unix 时间戳
* 记录 record：事件内容本身，JSON 格式

比如一条 Apache 的访问日志：

192.168.0.1 - - [28/Feb/2013:12:00:00 +0900] "GET / HTTP/1.1" 200 777

通过 in\_tail 输入插件处理后:

tag: apache.access # 通过配置文件指定

time: 1362020400 # 28/Feb/2013:12:00:00 +0900

record: {"user": "-", "method": "GET", "code": 200, "size": 777, "host": "192.168.0.1", "path": "/"}

当 Fluentd 收到一条事件后会经过一系列的处理流程：

* 修改事件的相关字段
* 过滤掉一些不需要的事件
* 路由事件输出到不同的地方

## Kibana: 日志展示和分析

Kibana 是 Elasticsearch 的一个功能强大的数据可视化 Dashboard，Kibana 允许你通过 web 界面来浏览 Elasticsearch 日志数据.

## K8S部署EFK

namespace/

namespace.yml

logging/

elastic-search/

service.yml

statefulset.yml

kibana/

service.yml

deployment.yml

$ kubectl create -f namespace/namespace.yml

$ kubectl get ns

$ kubectl create -f logging/elastic-search/service.yml

$ kubectl get svc -n kube-logging

$ kubectl create -f logging/elastic-search/statefulset.yml

$ kubectl get sts -n kube-logging

$ kubectl get po -n kube-logging

$ kubectl create -f logging/kibana/service.yml

$ kubectl create -f logging/kibana/deployment.yml

$ kubectl get pods -n kube-logging

// 创建命名空间，在其中安装所有日志相关的资源对象

**namespaces/namespace.yml**

kind: Namespace

apiVersion: v1

metadata:

name: kube-logging

// 创建elasticsearch服务

**logging/elastic-search/service.yml**

kind: Service

apiVersion: v1

metadata:

name: elasticsearch

namespace: kube-logging

labels:

app: elasticsearch

spec:

selector:

app: elasticsearch

clusterIP: None

ports:

- port: 9200 // 用于与REST API交互

name: rest

- port: 9300 // 用于节点间通信

name: inter-node

// 为 Pod 分配一个稳定的标识和持久化存储

**logging/elastic-search/statefulset.yml**

---

apiVersion: v1

kind: PersistentVolume

metadata:

name: es-cluster-pv

namespace: kube-logging

labels:

app: elasticsearch

spec:

capacity:

storage: 100Gi

accessModes:

- ReadWriteMany

storageClassName: logstorage

persistentVolumeReclaimPolicy: Retain

hostPath:

path: "/var/simulation-development-builder/kube-logging-elastic-search-service"

type: DirectoryOrCreate

---

apiVersion: apps/v1

kind: StatefulSet

metadata:

name: es-cluster

namespace: kube-logging

spec:

serviceName: elasticsearch

replicas: 1 // 副本数1

selector:

matchLabels:

app: elasticsearch

template:

metadata:

labels:

app: elasticsearch

spec:

containers:

- name: elasticsearch

image: "xxx-ehv.ta.philips.com/elasticsearch/elasticsearch:7.16.3"

securityContext:

allowPrivilegeEscalation: false

readOnlyRootFilesystem: false

privileged: false

resources:

limits:

cpu: 1000m

requests:

cpu: 100m

ports:

- containerPort: 9200

name: rest

protocol: TCP

- containerPort: 9300

name: inter-node

protocol: TCP

volumeMounts:

- name: data

mountPath: /usr/share/elasticsearch/data

- name: es-http-tls

mountPath: /usr/share/elasticsearch/config/certs

readOnly: true

- name: elasticsearch-config

mountPath: /usr/share/elasticsearch/config/elasticsearch.yml

subPath: elasticsearch.yml

env:

- name: cluster.name

value: k8s-logs

- name: node.name

valueFrom:

fieldRef:

fieldPath: metadata.name

- name: discovery.seed\_hosts

value: "es-cluster-0.elasticsearch"

- name: cluster.initial\_master\_nodes

value: "es-cluster-0"

- name: ES\_JAVA\_OPTS

value: "-Xms512m -Xmx512m"

- name: ELASTIC\_PASSWORD

valueFrom:

secretKeyRef:

name: elasticsearch-secret

key: RT\_ELASTICSEARCH\_PASSWORD

- name: logger.org.elasticsearch

value: "ERROR"

volumes:

- name: elasticsearch-config

configMap:

name: elasticsearch-config

- name: es-http-tls

secret:

secretName: elasticsearch-certificates-secret

initContainers:

- name: fix-permissions

image: radonc-docker-virtual.artifactory-ehv.ta.philips.com/busybox:1.31.1

command: ["sh", "-c", "chown -R 1000:1000 /usr/share/elasticsearch/data"]

securityContext:

privileged: true

volumeMounts:

- name: data

mountPath: /usr/share/elasticsearch/data

- name: increase-vm-max-map

image: radonc-docker-virtual.artifactory-ehv.ta.philips.com/busybox:1.31.1

command: ["sysctl", "-w", "vm.max\_map\_count=262144"]

securityContext:

privileged: true

- name: increase-fd-ulimit

image: radonc-docker-virtual.artifactory-ehv.ta.philips.com/busybox:1.31.1

command: ["sh", "-c", "ulimit -n 65536"]

securityContext:

privileged: true

volumeClaimTemplates: //定义持久化模板，为 Pod 创建 PersistentVolume 100G

- metadata:

name: data

labels:

app: elasticsearch

spec:

accessModes:

- "ReadWriteMany"

storageClassName: logstorage

resources:

requests:

storage: 100Gi

**logging/kibana/service.yml**

apiVersion: v1

kind: Service

metadata:

name: kibana

namespace: kube-logging

labels:

application: kibana

spec:

type: ClusterIP

selector:

application: kibana

ports:

- name: kibana-service

protocol: TCP

port: 5601

targetPort: 5601

**logging/kibana/deployment.yml**

apiVersion: apps/v1

kind: Deployment

metadata:

name: kibana

namespace: kube-logging

labels:

application: kibana

spec:

replicas: 1

selector:

matchLabels:

application: kibana

template:

metadata:

namespace: kube-logging

labels:

application: kibana

spec:

containers:

- name: kibana

image: "radonc-docker-virtual.artifactory-ehv.ta.philips.com/kibana/kibana:7.12.0"

securityContext:

allowPrivilegeEscalation: false

readOnlyRootFilesystem: false

privileged: false

resources:

limits:

cpu: 1000m

requests:

cpu: 100m

readinessProbe:

initialDelaySeconds: 60

periodSeconds: 5

timeoutSeconds: 1

successThreshold: 1

failureThreshold: 1

tcpSocket:

port: 5601

env:

- name: ELASTICSEARCH\_URL

value: "https://elasticsearch:9200"

- name: ELASTICSEARCH\_HOSTS

value: "https://elasticsearch.kube-logging.svc.cluster.local:9200"

- name: XPACK\_SECURITY\_ENABLED

value: "true"

- name: ELASTICSEARCH\_USERNAME

valueFrom:

secretKeyRef:

name: elasticsearch-secret

key: RT\_ELASTICSEARCH\_USERNAME

- name: ELASTICSEARCH\_PASSWORD

valueFrom:

secretKeyRef:

name: elasticsearch-secret

key: RT\_ELASTICSEARCH\_PASSWORD

- name: LOGGING\_QUIET

value: "true"

volumeMounts:

- name: config

mountPath: /usr/share/kibana/config/kibana.yml

readOnly: true

subPath: kibana.yml

- name: es-http-tls

mountPath: /usr/share/kibana/config/certs/es

readOnly: true

- name: kibana-certs

mountPath: /usr/share/kibana/config/certs/kibana

readOnly: true

ports:

- containerPort: 5601

volumes:

- name: config

configMap:

name: kibana-config

- name: es-http-tls

secret:

secretName: elasticsearch-certificates-secret

- name: kibana-certs

secret:

secretName: kibana-certificates-secret

# VDI (virtual desktop infrastructure): ETX, FastX

**X Windows 终端仿真程序 (**远程应用程序访问解决方案**)**

X Window是一个分层的架构，它分为Serve和Client。X Server负责用户的输入和图形界面的显示，而Client程序需要连接到X Server，然后请求**X Server绘制图形界面**，同时从X Server接受用户的输入。

在同一台机器上完全可以运行多个X Server。在X Window中，可以通过hostname: display\_number. screen\_number来指定一个屏幕。display相当于是计算机配备的一套输入输出设备，默认是display :0。/usr/bin/X都是X Server程序

$sudo X :1 -retro # 在display :1上运行一个X Server

XWindows仿真器：Exceed OnDemand, ETX (Exceed VA TurboX), RealVNC, 以及NoMachine的NX

Virtual applications and virtual desktops: for remote apps and remote desktops

Remote sound and video are streamed to the local device and USB redirection allows for local printing of remote content. Using the latest technology graphics card through server-side rendering delivers superior rendering performance, with rendering results transmitted to a user’s computer or tablet.

Connected multiple systems, integration of components

Table

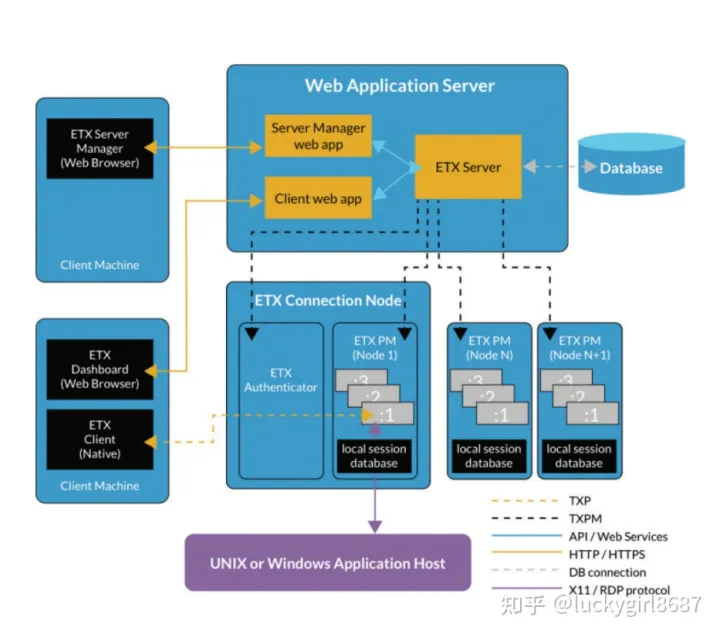
Description automatically generatedTable

Description automatically generated

* **ETX (OpenText Exceed TurboX):**

<https://www.opentext.com/products/exceed-turbox>

OpenText Exceed (1990年) -> OpenText Exceed onDemand (2000 年) 。OpenText Exceed TurboX 是一种用于**桌面虚拟化和远程访问企业应用程序和数据**的高级解决方案，可满足现代企业的需求，尤其是那些具有混合 UNIX®/Microsoft® Windows® 托管环境的企业。



ETX 服务器：是一个在 Apache® Tomcat® Web 应用程序服务器上运行的 Web 应用程序，并在用户首次连接到 ETX Web 门户时呈现给用户。 **ETX 服务器向用户显示仪表板**或向管理员显示服务器管理器。 **用户**可以使用 Web Dashboard 启动新会话、创建配置文件或管理现有会话。 **管理员**使用服务器管理器来监控和管理解决方案的各个方面，从用户注册到生成许可证和资源使用报告。

ETX 连接节点：负责启动会话代理（管理和压缩远程会话的应用程序）

ETX 客户端运行时：ETX 依靠本地客户端运行时来处理远程通信和交互。

当用户启动远程应用程序或桌面时，其中一个 ETX 连接节点会创建一个进程（称为“代理”）以充当用户和远程主机之间的中介。 **代理进程负责与主机（作为会话客户端）交互并压缩远程显示以通过 WAN 传输给最终用户。**

FastX

* Desktop Client桌面客户端 + Browser Client浏览器客户端

通过浏览器（Firefox, Chrome, Internet Explorer, Safari等）连接远程Linux台式机和应用程序上。在浏览器网址区域输入Linux服务器的名称。FastX用户界面便会出现，你可以开始新的会话，或者继续一个已经在远程主机上运行的被推迟的会话。

* FastX在一个高度精简的协议上运行，这使其性能前所未有地超越了局域网和广域网连接。
* 持续性。用户也可以推迟他们的会话，断开他们的个人电脑连接，晚些时候或者从另一台个人电脑上继续对话。

# Tools

## Fortify

## BlackDuck

## Nexus

# Drools

**introduction**

is a Business Logic integration Platform (BLiP)

Drools is split into two main parts: Authoring and Runtime.

Authoring: Authoring process involves the creation of Rules files (.DRL files).

Runtime: It involves the creation of working memory and handling the activation.

Rules are pieces of knowledge often expressed as, "When some conditions occur, then do some tasks."

Logic and Data Separation

The data resides in the Domain Objects and the business logic resides in the Rules.

何时使用规则引擎？

并非所有应用程序都应使用规则引擎。如果业务逻辑代码包括很多 if-else 语句，则应考虑使用一个规则引擎。**维护复杂的 Boolean 逻辑**可能是非常困难的任务，而规则引擎可以帮助您组织该逻辑。当您可以使用声明方法而非命令编程语言表达逻辑时，变化引入错误的可能性会大大降低。

对客户的了解也是该决策的一个因素。在开发周期期间甚至部署之后添加和更改业务逻辑需求的倾向。

使用规则引擎场景特点：

单条规则相对简单

规则数量相对庞大

规则之间会有冲突

一条规则本身会触发另一条规则

规则有可能会产生变动（很多时候可能唯一需要变动的就是规则）

有些逻辑需要的参数我们并不能定义在规则中，而是在数据库表中进行配置。因此我们常见的业务逻辑层的开发，并不能先设计出一个数据模型，然后再在此基础上抽象逻辑。

因此我们发现Drools等规则引擎很难用

规则引擎:严格来说，它是一种嵌入到应用程序中的一个组件，能很好的把业务决策从应用程序框架中分离出来，然后使用预定义的方言（dialect）编写语义模块和业务决策模块，使用约定好的语法规范，接受用户的输入，然后解析用户的业务规则，然后根据解析好的业务规则，作出业务决策。

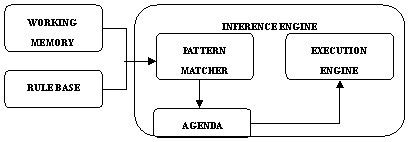
Drools是基于Java的规则引擎框架，是JBoss开源社区中的一个为Java量身定制的、基于RETE算法的产生式规则引擎的实现。大致的工作原理是，基于XML、DRL（Drools规则配置文件）的基础上，通过一个内置的解析器，把业务规则翻译成AST（Abstract Syntax Tree），最终会映射编译成Java的代码包，然后在程序运行的时候，加载这些代码包中的业务规则，并把在工作内存空间的规则和事实进行匹配，看下事实是否符合业务规则的约定

规则引擎与工作流引擎概念

业务流程执行语言(BPEL)

中央规则信息库

基于规则的专家系统（RBES）包括三部分：Rule Base（knowledge base）、Working Memory（fact base）和Inference Engine



工作流引擎现在演化为 业务过程管理，最基本的概念还是业务过程的流转，

业务规则侧重业务逻辑的控制，比如零售行业的价格规则，它会针对 客户 商品/品类 地理位置等设计一个非常复杂的价格管理体系；与业务规则非常紧密的是事件引擎，比如政府部门的应急处理系统

规则引擎使用了匹配规则的方式来进行，因此在应用这些规则引擎时。首先需要将我们具体应用中的业务逻辑做抽象，抽象成一条条规则之后，再打包成一个规则包。一个规则包相当于一个智能块。当数据传递给这个智能块后，系统会以匹配的方式应用满足条件的逻辑处理。

Drools is a Business Rules Management System (BRMS) solution

Drools Workbench (web UI for authoring and management) + Drools Expert (business rules engine)

https://www.drools.org/#

应用程序中业务逻辑最常见的方法是编写 Java 代码来实现需求文档的规则和逻辑

规则引擎试图降低应用程序业务逻辑的开发和维护的困难。可以将**规则引擎看作实现复杂业务逻辑的框架**。大多数规则引擎允许您使用声明性编程来表达对于某些给定信息或知识有效的结果

Drools 是用 Java 语言编写的开放源码规则引擎, Drools允许使用声明方式表达业务逻辑。可以使用非 XML 的本地语言编写规则

<http://blog.csdn.net/quzishen/article/details/6163012/>

接口：传参数，接口实现执行，获取结果

drools: 传递数据，规则检查和执行，获取结果

在drools,传递数据称为fact对象（java bean),当一个java bean插入到workingMemory中，规则使用的是原有对象的引用，规则通过对fact对象的读写，实现对应用数据的读写，对于其中的属性，需要提供getter setter访问器，规则中，可以动态的往当前workingMemory中插入删除新的fact对象。

规则文件: .drl (drools rule language)

Drl中可以通过Import的方式引入Model类，也可以调用Java的各种函数，也可以自己定义Class and Function

Drl

package package-name

imports

globals

functions

queries

rules

Rules就是规则的部分，结构如下：

rule "name"

attributes

when

LHS(The Rule Language)

then

RHS(Java, Pthyon, Groovy)

End

每一条规则都有名称

每一条规则都有属性, 定义当前规则执行的一些属性等，比如是否可被重复执行、过期时间、生效时间等。

When定义当前规则的条件，等同于if里的条件判断，使用Drools自己的语法规则

Then当前规则条件满足后执行的操作,等同于if里的执行语句，可以写普通java代码，也可以直接调用Fact对象的方法来操作应用

***属性：***

**no-loop true**: 定义当前的规则是否不允许多次循环执行，默认是false

**lock-on-active true**：通过这个标签，可以控制当前的规则只会被执行一次，因为一个规则的重复执行不一定是本身触发的，也可能是其他规则触发的，所以这个是no-loop的加强版

**date-expires**：设置规则的过期时间，如"2011-01-31 23:59:59"

**date-effective**：设置规则的生效时间

**duration**：规则定时，duration 3000 3秒后执行规则

**salience**：优先级，数值越大越先执行，这个可以控制规则的执行顺序。

***条件：***

当前规则只有在条件都匹配的时候才会执行

Drools提供了十二中类型比较操作符：

**> >= < <= == != contains / not contains / memberOf / not memberOf /matches/ not matches**

如：

when

$customer:Customer()

$message:Message(status==0 || (status > 1 && status <=100))

当前规则只有在这三个条件都匹配的时候才会执行RHS部分

$message:Message(status==0 || (status > 1 && status <=100))：当前的workingMemory存在status为0或在[1 100]范围内的Ｍessage对象，这个对象通常是通过外部java代码插入或者自己在前面已经执行的规则的RHS部分中insert进去的。$message代表着当前条件的引用变量，在后续的条件部分和RHS部分中，可以使用当前的变量去引用符合条件的FACT对象，修改属性或者调用方法等

如果条件全部是 &&关系，可以使用“,”来替代

***执行：***

**调用Fact的方法：**如 $message.execute();操作数据库等等一切操作

调用Drools API方法：

**insert**：往当前workingMemory中插入一个新的Fact对象

**update**：更新

**modify**：修改

**retract**：删除

insert update modify and restract会触发规则的再次执行，除非使用no-loop限定；

即停止执行剩下的规则，重新执行所有规则

**调用规则文件自定义方法和类：**

function void console {

System.out.println();

StringUtils.getId(); // 调用外部静态方法

}

declare Address

@author(quzishen) // 元数据，仅用于描述信息

@createTime(2011-1-24)

city : String @maxLengh(100)

postno : int

end

Address address = new Address();

**setup**

<https://nheron.gitbooks.io/droolsonboarding/content/>

Drools 环境: eclipse + Drools Engine + Drools and jBPM tools

<http://blog.csdn.net/yeomer/article/details/54291263>

KieServices就是一个中心，通过它来获取的各种对象来完成规则构建、管理和执行等操作。

KieContainer就是一个KieBase的容器

KieBase就是一个知识仓库，包含了若干的规则、流程、方法等

KieSession就是一个跟Drools引擎打交道的会话，其基于KieBase创建，它会包含运行时数据，包含“事实 Fact”，并对运行时数据事实进行规则运算

**Drools doc**

Guvnor is a BRMS=Business Rule Management System　in versions 5.X ->

kie Workbench in versions 6.x till 6.3 ->

Business central starting with version 6.4

business central : the BRMS (Business Rule Management System) and the BPMS (Business Process Management System

Guided Decision Tree

Guided Decision Table

Decision tables are not recommended

for rules that do not follow a set of templates, or where there are a small number of rules (or if there is a dislike towards software like Excel or OpenOffice.org).

By convention the second column ("B") is used for this, but it can be any column (convention is to leave a margin on the left for notes).Everything to the left of this is ignored.

|  |  |  |
| --- | --- | --- |
|  | RuleSet | The package name |
|  | Sequential | true |
|  | Import | Java package |
|  |  |  |
|  |  |  |

if the plug-in is being used (Rule Workbench IDE), the wizard can generate a spreadsheet from a template (to edit it an xls compatible spreadsheet editor will need to be used)

Decision tables lend themselves to close collaboration between IT and domain experts, while making the business rules clear to business analysts, it is an ideal separation of concerns.

With rule templates the data is completely separate from the rules

Guided Rule

drools workbench

drools workbench docker

<https://hub.docker.com/r/jboss/drools-workbench/>

<https://hub.docker.com/r/jboss/drools-workbench-showcase/>

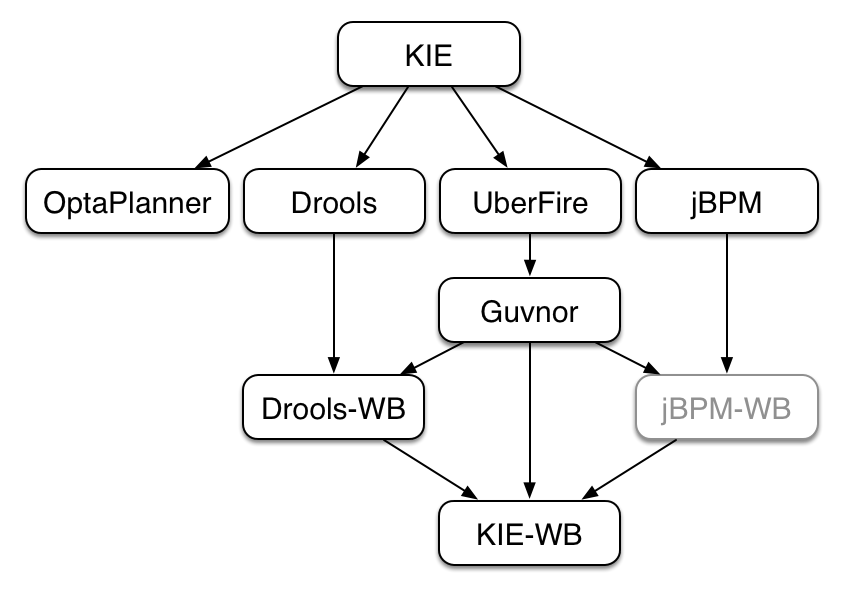
$docker run -p 8080:8080 -p 8001:8001 -d --name drools-workbench jboss/drools-workbench-showcase:latest

access URL: http://localhost:8080/drools-wb

USER PASSWORD ROLE

admin admin admin,analyst,kiemgmt

Drools Documents



KIE (Knowledge Is Everything) = Drools + jBPM

Authoring of knowledge using a UI metaphor, such as: DRL, BPMN2, decision table, class models.

User interaction with the KieSession, via command line or UI.

System interaction with the KieSession, via API.

The loading of a JAR to provide a KIE session (KieSession), for which the application can interact with.

KIE exposes the JAR at runtime via a KIE container (KieContainer).

KieSessions, for the runtime's to interact with, are created from the KieContainer.

KIE will scan the classpath to find all the JARs with a kmodule.xml in it.

A Kie Project = Maven project + kmodule.xml (from which the KieBases and KieSessions that can be created)

all the Java sources and the Kie resources are compiled and deployed into the KieContainer which makes its contents available for use at runtime

kmodule.xml configure the KieBase(s) and KieSession(s)

**KieBase contains all the application's knowledge definitions such as rules, processes, functions, and type models**

KieSession are created from the KieBase into which data can be inserted and from which process instances may be started, the **KieSession stores and executes on the runtime data**. It is created from the KieBase or more easily can be created directly from the KieContainer if it has been defined in the **kmodule.xml** file

<kmodule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns="http://www.drools.org/xsd/kmodule">

<kbase name="KBase1" default="true" declarativeAgenda="enabled" packages="org.domain.pkg1">

　<**ksession** name="KSession2\_1" type="stateful" default="true"/>

　　<ksession name="KSession2\_2" type="stateless" default="false" />

</kbase>

</kmodule>

\*.java

//KieServices access all the Kie building and runtime facilities

KieServices kieServices = KieServices.Factory.get();

//KieContainer reads the files to be built from the classpath

KieContainer kContainer = kieServices.getKieClasspathContainer();

//retrieve the KieBases and KieSessions from the KieContainer using their names.

KieBase kBase1 = kContainer.getKieBase("KBase1");

KieSession kieSession1 = **kContainer.newKieSession**("KSession2\_1");

StatelessKieSession kieSession2 = kContainer.newStatelessKieSession("KSession2\_2");

RUNNING

This FactHandle is the token used to represent your inserted object within the WorkingMemory. It is also used for interactions with the WorkingMemory when you wish to delete or modify an object

FactHandle stiltonHandle = ksession.insert( stilton );

stilton.setPrice( 100 );

workingMemory.update( stiltonHandle, stilton );

ksession.delete( stiltonHandle );

Query

Queries are used to retrieve fact sets based on patterns, as they are used in rules.

Agenda

When a rule is fully matched a Match is created, referencing the rule and the matched facts, and placed onto the Agenda. The Agenda controls the execution order of these Matches using a Conflict Resolution strategy.

The default conflict resolution strategies employed by Drools are: Salience and LIFO (last in, first

out).

when a flow is needed a number of possibilities exist, including but not limited to: agenda groups, rule flow groups, activation groups, control/semaphore facts

ActivationGroup: An activation group is a set of rules bound together by the same "activation-group" rule attribute. In this group only one rule can fire, and after that rule has fired all the other rules are cancelled from the agenda

RuleFlowGroup: A rule flow group is a group of rules associated by the "ruleflow-group" rule attribute. These rules can only fire when the group is activate. The group itself can only become active when the elaboration of the ruleflow diagram reaches the node representing the group

AgendaGroup: Agenda groups are known as "modules" in CLIPS terminology, If your rules have a clear need for multiple "phases" or "sequences" of processing, consider using agenda-groups for this purpose.

Event Model

notified of rule engine events, including rules firing, objects being asserted, etc.This allows you, for instance, to separate logging and auditing activities from the main part of your application (and the rules).

Rule Execution Modes

Passive Mode is most suitable for Rule Engine applications which need to explicitly control when the engine shall evaluate and fire the rules, or for CEP applications making use of the Pseudo Clock. Active Mode is most effective for Rule Engine applications which delegate control of when rules are evaluated and fired to the engine, or for typical CEP application making use of the Real Time Clock.

Drools offers a fireUntilHalt() method, that starts the engine in Active Mode, which is asynchronous

in behavior, where rules will be continually evaluated and fired, until a halt() call is made.

new Thread( new Runnable() {

@Override

public void run() {

session.fireUntilHalt();

}

} ).start();

session.insert( tick1 );

... Thread.sleep( 1000L ); ...

session.insert( tick2 );

... Thread.sleep( 1000L ); ...

session.insert( tick3 );

session.halt();

Rules file

package package-name

imports

globals

functions

queries

rules

Global

Drools automatically imports classes from the Java package of the same name, and also from the package java.lang.

With global you define global variables. They are used to make application objects available to the rules. Typically, they are used to provide data or services that the rules use, especially application services used in rule consequences, and to return data from the rules, like logs or values added in rule consequences, or for the rules to interact with the application, doing callbacks. Globals are not inserted into the Working Memory, and therefore a global should never be used to establish conditions in rules except when it has a constant immutable value. The engine cannot be notified about value changes of globals and does not track their changes

It is a best practice to set all global values before asserting any fact to the working memory.

List list = new ArrayList();

KieSession kieSession = kiebase.newKieSession();

kieSession.setGlobal( "myGlobalList", list );

Globals are not designed to share data between rules and they should never be used for that purpose. Rules always reason and react to the working memory state, so if you want to pass data from rule to rule, assert the data as facts into the working memory

Function

The main advantage of using functions in a rule is that you can keep the logic all in one place, and you can change the functions as needed (which can be a good or a bad thing).

declare Address

number : int

streetName : String

city : String

end

import java.util.Date

declare Person

name : String

dateOfBirth : Date

address : Address

end

declare enum DaysOfWeek

SUN,MON,TUE,WED,THU,FRI,SAT;

end

When you declare a new fact type, Drools will, at compile time, generate bytecode that implements a Java class representing the fact type

Declared types are usually used inside rules files, while Java models are used when sharing the model between rules and applications

Traits

The same fact may have multiple dynamic types which do not fit naturally in a class hierarchy. Traits allow to model this very common scenario. A trait is an interface

import org.drools.core.factmodel.traits.Traitable;

declare Customer

@Traitable

code : String

balance : long

end

Rule

rule "<name>"

<attribute>\*

when

<conditional element>\*

then

<action>\*

End

no-loop

type: Boolean, default value: false

Setting no-loop to true will skip the creation of another Activation for the rule with the current set of facts.

ruleflow-group

type: String, default value: N/A

Rules that are assembled by the same ruleflow-group identifier fire only when their group is active.

agenda-group

type: String, default value: MAIN

Only rules in the agenda group that has acquired the focus are allowed to fire.

auto-focus

type: Boolean, default value: false

activation-group

type: String, default value: N/A

Rules that belong to the same activation-group, identified by this attribute's string value, will only fire exclusively

lock-on-active

type: Boolean, default value: false

It's

ideal for calculation rules where you have a number of rules that modify a fact and you don't want any rule re-matching and firing again.

salience

type: integer, default value: 0

dynamic salience where you can use an expression involving bound variables.

rule "Fire in rank order 1,2,.."

salience( -$rank)

when

Element( $rank : rank,...)

then

...

date-effective

type: String, containing a date and time definition, default value: N/A

A rule can only activate if the current date and time is after date-effective attribute.

date-expires

type: String, containing a date and time definition, default value: N/A

A rule cannot activate if the current date and time is after the date-expires attribute.

duration

type: long, default value: no default value

The duration dictates that the rule will fire after a specified duration, if it is still true

Timers and Calendars

rule "Send SMS every 15 minutes"

timer (cron:\* 0/15 \* \* \* ?)

when

$a : Alarm( on == true )

then

channels[ "sms" ].insert( new Sms( $a.mobileNumber, "The alarm is still on" );

end

Cron (indicated by "cron:") timers follow standard Unix cron expressions

timer (int: 30s 10s; start=3-JAN-2010, end=5-JAN-2010) An Interval Timer with a start and an end

rule "weekdays are high priority"

calendars "weekday"

timer (int:0 1h)

when

Alarm()

then

send( "priority high - we have an alarm" );

end

rule "weekend are low priority"

calendars "weekend"

timer (int:0 4h)

when

Alarm()

then

send( "priority low - we have an alarm" );

end

LHS

Any method executed on a fact in the LHS should be a read only method.

If the LHS is empty, it will be considered as a condition element that is always true

Property access on Java Beans (POJO's)

Person( age == 50 )

//first check getAge(), then check age()

Person( address.houseNumber == 50 ) <=> Person( getAddress().getHouseNumber() == 50 )

Java expression

Person( age > 100 && ( age % 10 == 0 ) )

Person( Math.round( weight / ( height \* height ) ) < 25.0 )

The == operator has null-safe equals() semantics:

The != operator has null-safe !equals() semantics:

尽量用’,’表示&&, 若条件复杂，就用&&, ||, ==, !=

Person( name == "mark", address.( city == "london", country == "uk") ) 内嵌对象address

Cheese( bestBefore < "27-Oct-2009" )

Person( childList[0].age == 18 )

Person( credentialMap["jsmith"].valid )

Person( age ( (> 30 && < 40) || (> 20 && < 25) ) )

or Date fields, < means before, for String fields, it means alphabetically lower.

Person( $streetName : address!.street )

等价于 Person( address != null, $streetName : address.street )

The !. operator allows to derefencing in a null-safe way. 等价于java的Optional<T>

CheeseCounter( cheeses contains "stilton" )

CheeseCounter( cheeses not contains "cheddar" )

CheeseCounter( cheese memberOf $matureCheeses )

Cheese( name soundslike 'foobar') // match cheese "fubar" or "foobar" based on the Soundex algorithm

Message( routingValue str[startsWith] "R1" )

Message( routingValue str[endsWith] "R2" )

Message( routingValue str[length] 17 )

Person( $cheese : favouriteCheese )

Cheese( type in ( "stilton", "cheddar", $cheese ) ) )

and, or, not, exists, forall, from, collect, accumulate

rule "validate zipcode"

when

$p : Person( )

$a : Address( zipcode == "23920W") from $p.address

then

rule "apply 10% discount to all items over US$ 100,00 in an order"

when

$order : Order()

$item : OrderItem( value > 100 ) from $order.items

Then

java.util.ArrayList

rule "Raise priority if system has more than 3 pending alarms"

when

$system : System()

$alarms : ArrayList( size >= 3 ) from collect( Alarm( system == $system, status == 'pending' ) )

then

the rule will look for all pending alarms in the working memory for each given system and group them in ArrayLists. If 3 or more alarms are found for a given system, the rule will fire.

java.util.LinkedList;

rule "Send a message to all mothers"

when

$town : Town( name == 'Paris' )

$mothers : LinkedList()

from collect( Person( gender == 'F', children > 0 )

from $town.getPeople()

)

then

Accumulate allows a rule to iterate over a collection of objects, executing custom actions for each of the elements, and at the end, it returns a result object

rule "Raise alarm"

when

$s : Sensor()

accumulate( Reading( sensor == $s, $temp : temperature );

$min : min( $temp ),

$max : max( $temp ),

$avg : average( $temp );

$min < 20, $avg > 70 )

then

// raise the alarm

End

eval

the best practice is to add it as the last conditional element in the LHS of a rule.

ideal for being used when functions return values that change over time, which is not allowed within Field Constraints.

when

p1 : Parameter()

p2 : Parameter()

eval( p1.getList().containsKey( p2.getItem() ) )

RHS

If you find you need imperative and/or conditional code in the RHS, then maybe you should be breaking that rule down into multiple rules, The main purpose of the RHS is to insert, delete or modify working memory data.

insert(object);

update(object);

rule "modify stilton"

when

$stilton : Cheese(type == "stilton")

then

**modify**( $stilton ){

setPrice( 20 ),

setAge( "overripe" ) }

end

delete(object);

drools.halt()

drools.getWorkingMemory()

drools.setFocus( String s) sets the focus to the specified agenda group.

drools.getRule().getName()

drools.getTuple()

drools.getActivation()

drools.setFocus( "CleanUp" )

等价于kcontext.getKieRuntime().getAgenda().getAgendaGroup( "CleanUp" ).setFocus();

drools.getKieRuntime().getQueryResults(...)

getKieBase()

setGlobal(...), getGlobal(...)

getEnvironment()

A query is a simple way to search the working memory for facts that match the stated conditions.

Therefore, it contains only the structure of the LHS of a rule. To return the results use ksession.getQueryResults("name"), where "name" is the query's name.

Query People over the age of x, and who live in y

query "people over the age of x" (int x, String y)

person : Person( age > x, location == y )

end

QueryResults results = ksession.getQueryResults( "people over the age of 30" );

System.out.println( "we have " + results.size() + " people over the age of 30" );

System.out.println( "These people are are over 30:" );

for ( QueryResultsRow row : results ) {

Person person = ( Person ) row.get( "person" );

System.out.println( person.getName() + "\n" );

}

Queries can now call other queries, this combined with optional query arguments provides derivation query style backward chaining.

declare Location

thing : String

location : String

end

query isContainedIn( String x, String y )

Location(x, y;) or ( Location(z, y;) and ?isContainedIn(x, z;) )

end

query checkLength(String $s, int $l)

$s := String( length == $l )

end

rule CheckPersonNameLength

when

$i : Integer()

$p : Person()

checkLength( $p.name, 1 + $i + $p.age; )

then

end

Domain Specific Languages

If your rules need to be read and validated by domain experts (such as business analysts, for instance) who are not programmers, you should consider using a DSL; it hides implementation details and focuses on the rule logic proper

DSLs have no impact on the rule engine at runtime, they are just a compile time feature, requiring a special parser and transformer

Given a DSL, you write rules in DSL rule (or DSLR) files, which will be translated into DRL files.

authors using the DSL should still be able to identify DSL phrases by some fixed text.

The DSL definitions

[when]There is a Cheese with=Cheese()

[when]- age is less than {age}=age<{age}

[when]- type is '{type}'=type=='{type}'

[when]- country equal to '{country}'=country=='{country}'

write rules

There is a Cheese with

- age is less than 42

- type is 'stilton'

DRL result of parser

Cheese(age<42, type=='stilton')

DSL definitions

[when][]is less than or equal to=<=

[when][]is less than=<

[when][]is greater than or equal to=>=

[when][]is greater than=>

[when][]is equal to===

[when][]equals===

[when][]There is a Cheese with=Cheese()

[when][]- {field:\w\*} {operator} {value:\d\*}={field} {operator} {value}

write rules

There is a Cheese with

- age is less than 42

- rating is greater than 50

- type equals 'stilton'

DRL result of parser

Cheese(age<42, rating > 50, type=='stilton')

Drools Fusion, Drools Flow

• Business Rules Management

• Business Processes Management

• Complex Event Processing

Complex Event Processing (CEP)

Event is the record of the change of a particular piece of data in the domain.

CEP scenarios share several common and distinguishing characteristics:

In CLOUD mode, the engine assumes that all facts and events are known in advance (there is no concept of flow of time)

in STREAM mode, negative patterns with temporal constraints may require the engine to wait for a time period before activating a rule

//马上触发

rule "Sound the alarm"

when

$f : FireDetected( )

not( SprinklerActivated( ) )

then

// sound the alarm

end

//时间控制

rule "Sound the alarm"

duration( 10s )

when

$f : FireDetected( )

not( SprinklerActivated( this after[0s,10s] $f ) )

then

// sound the alarm

end

Reasoning over time requires a reference clock

example, if a rule reasons over the average price of a given stock over the last 60 minutes, how the engine knows what stock price changes happened over the last 60 minutes in order to calculate the average?

//时间窗

when

TemperatureThreshold( $max: max )

Number( doubleValue > $max)

from accumulate(

SensorReading( $temp : temperature ) **over window:time( 10m )**,

average($temp) )

//长度窗

SensorReading( $temp : temperature ) over window:length( 100 )

Drools generalized the concept of a stream as an "entry point" into the engine. An entry point is for drools a gate from which facts come. The facts may be regular facts or special facts like events.

Entry points are declared implicitly in Drools by directly making use of them in rules

when

WithdrawRequest( $ai : accountId, processed == true ) from entry-point "Branch Stream"

CheckingAccount( accountId == $ai )

then

// apply a $2 fee on

EntryPoint atmStream = session.getEntryPoint( "ATM Stream" );

// and start inserting your facts into the entry point

atmStream.insert( aWithdrawRequest );

Temporal Reasoning

$eventA : EventA( this after[ 3m30s, 4m ] $eventB )

等价于

3m30s <= $eventA.startTimestamp - $eventB.endTimeStamp <= 4m

$eventA : EventA( this before[ 3m30s, 4m ] $eventB )

$eventA : EventA( this coincides[15s, 10s] $eventB ) 事件A与事件B发生很接近

等价于

abs( $eventA.startTimestamp - $eventB.startTimestamp ) <= 15s &&

abs( $eventA.endTimestamp - $eventB.endTimestamp ) <= 10s

$eventA : EventA( this during[ 2s, 6s, 4s, 10s ] $eventB )

等价于

2s <= $eventA.startTimestamp - $eventB.startTimestamp <= 6s &&

4s <= $eventB.endTimestamp - $eventA.endTimestamp <= 10s

$eventA : EventA( this finishes[ 5s ] $eventB )

等价于

$eventB.startTimestamp < $eventA.startTimestamp &&

abs( $eventA.endTimestamp - $eventB.endTimestamp ) <= 5s

$eventA : EventA( this finishedby $eventB )

等价于

$eventA.startTimestamp < $eventB.startTimestamp &&

$eventA.endTimestamp == $eventB.endTimestamp

$eventA : EventA( this includes $eventB )

等价于

$eventA.startTimestamp < $eventB.startTimestamp <= $eventB.endTimestamp < $eventA.endTimestamp

$eventA : EventA( this meets[ 5s ] $eventB )

等价于

abs( $eventB.startTimestamp - $eventA.endTimestamp) <= 5s

$eventA : EventA( this metby[ 5s ] $eventB )

等价于

abs( $eventA.startTimestamp - $eventB.endTimestamp) <= 5s

$eventA : EventA( this overlaps[ 5s ] $eventB )

等价于

$eventA.startTimestamp < $eventB.startTimestamp < $eventA.endTimestamp < $eventB.endTimestamp

&& 0 <= $eventA.endTimestamp - $eventB.startTimestamp <= 5s

$eventA : EventA( this overlappedby[ 5s, 10s ] $eventB )

等价于

$eventB.startTimestamp < $eventA.startTimestamp < $eventB.endTimestamp < $eventA.endTimestamp

&& 5s <= $eventB.endTimestamp - $eventA.startTimestamp <= 10s

$eventA : EventA( this starts[ 5s ] $eventB )

等价于

abs( $eventA.startTimestamp - $eventB.startTimestamp ) <= 5s &&

$eventA.endTimestamp < $eventB.endTimestamp

$eventA : EventA( this starts[ 5s ] $eventB )

等价于

abs( $eventA.startTimestamp - $eventB.startTimestamp ) <= 5s &&

$eventA.endTimestamp > $eventB.endTimestamp

Integration with Spring

Defining a file logger

<kie:kmodule **id**="loggers\_module">

<kie:kbase **name**="drl\_kiesample" **packages**="drl\_kiesample">

<kie:ksession **name**="ConsoleLogger-statefulSession" **type**="stateful">

<kie:fileLogger **id**="fl\_logger" **file**="#{ systemProperties['java.io.tmpdir'] }/log1"/>

<kie:fileLogger **id**="tfl\_logger" **file**="#{ systemProperties['java.io.tmpdir'] }/log2" **threaded**="true" **interval**="5"/>

</kie:ksession>

</kie:kbase>

</kie:kmodule>

<bean **id**="kiePostProcessor" **class**="org.kie.spring.KModuleBeanFactoryPostProcessor"/>

LoggerAdaptor adaptor = (LoggerAdaptor) context.getBean("fl\_logger");

adaptor.close();

例子1: 积分发放操作

https://wenku.baidu.com/view/b874963cf524ccbff0218402.html

发放积分可能伴随不同的运营策略和季节性调整，发放数目和规则完全不同，如果使用硬编码的方式去伴随业务调整而修改，代码的修改、管理、优化、测试、上线将是一件非常麻烦的事情，所以，将发放规则部分提取出来，交给Drools管理，可以极大程度的解决这个问题。

发放规则：

积分的发放参考因素有：交易笔数、交易金额数目、信用卡还款次数、生日特别优惠等。

定义规则：

1. 过生日，则加10分，并且将当月交易笔数翻倍后再计算积分 priority 1
2. 2011-某几个月，每月信用卡还款3次以上，每满3笔赠送30分
3. 当月购物总金额100以上，每100元赠送10分
4. 当月购物次数5次以上，每五次赠送50分
5. 特别的，如果全部满足了要求，则额外奖励100分
6. 发生退货，扣减10分
7. 退货金额大于100，扣减100分

**// fact**

**public** **class** Point {

**private** **boolean** isBirthday;

**private** **int** payNum;

**private** **double** buyAmount;

**private** **int** buyNum;

**private** **double** backAmount;

**private** **long** point;

}

**rules.point.AddPoint.drl**

**package** rules.point

**import** com.sample.point.Point;

**rule** "birth day"

**salience** 1

**lock-on-active** **true**

**when**

$p: Point(isBirthday == **true**)

**then**

$p.setPoint($p.getPoint()+10);

$p.setPayNum($p.getPayNum()\*2);

$p.setBuyAmount($p.getBuyAmount()\*2);

$p.setBuyNum($p.getBuyNum()\*2);

**update**($p);

**end**

**rule** "special month"

**date-effective** "1-April-2017"

**date-expires** "1-May-2017"

**when**

$p: Point(payNum >= 3)

**then**

$p.setPoint($p.getPoint() + $p.getPayNum()/3\*30);

**end**

**rule** "large amount"

**when**

$p: Point(buyAmount >= 100)

**then**

$p.setPoint($p.getPoint() + (**long**)$p.getBuyAmount()/100\*10);

**end**

**rule** "frequency buy"

**when**

$p: Point(buyNum >= 5)

**then**

$p.setPoint($p.getPoint() + $p.getBuyNum()/5\*50);

**end**

**rule** "large amount and frequency buy"

**when**

$p: Point(buyAmount>=100, buyNum>=5)

**then**

$p.setPoint($p.getPoint()+100);

**end**

**rules.point.SubstractPoint.drl**

**package** rules.point

**import** com.sample.point.Point;

**rule** " back"

**when**

$p: Point(backAmount>0)

**then**

$p.setPoint($p.getPoint()-10);

**end**

**rule** "large back"

**when**

$p: Point(backAmount>=100)

**then**

$p.setPoint($p.getPoint()-100);

**end**

**kmodule.xml**

**<**kbase name=*"rules"* packages=*"rules"*>

<ksession name=*"ksession-rules"*/>

</kbase>

<kbase name=*"point"* packages=*"rules.point"*>

<ksession name=*"ksession-rules-point"*/>

</kbase>

**Test.java**

// load up the knowledge base

KieServices ks = KieServices.Factory.*get*();

KieContainer kContainer = ks.getKieClasspathContainer();

KieSession kSession = kContainer.newKieSession("ksession-rules-point");

Point point = **new** Point();

point.setIsBirthday(**true**);

point.setPayNum(5);

point.setBuyAmount(500);

point.setBuyNum(5);

point.setBackAmount(100);

kSession.insert(point);

Message message = **new** Message();

message.setMessage("Hello World");

message.setStatus(Message.***HELLO***);

kSession.insert(message);

kSession.fireAllRules();

System.***out***.println("after rule");

System.***out***.println("the user's current point = " + point.getPoint());

注意：kSession　=> ksession-rules-point => rules.point => rules.point.AddPoint.drl, rules.point.SubstractPoint.drl

所以kSession workingMemeory里不论有什么fact,只能执行rules.point包下的规则文件

若要执行rules包下的规则文件，需要新开一个session

KieSession kSession = kContainer.newKieSession("ksession-rules");

kSession.insert(point);

kSession.insert(message);

kSession.fireAllRules();

当然把所有的规则放在一个包里面，session塞进所有的fact，从事执行包下面的所有规则文件，满足条件就触发规则

Example

例子2: Li-RADS

例子3: spring boot + drools

<https://scattercode.co.uk/2015/02/06/a-minimal-spring-boot-drools-web-service/>

<https://github.com/gratiartis/buspass-ws>