Handlers

Идеология от dvolodoin (2015-11-13):

> *логика такова*
>
> *есть callable*
>
> *когда он загружен, у него есть id*
>
> *callable -- это объект питона, который можно вызвать как функцию и он вернет результат*
>
> *вот два примера*
>
> *def callable1(a, b):*
>
> *return a + b*
>
> *второй вариант*
>
> *class callable2:*
>
> *def __call__(self, a, b):*
>
> *return a + b*
>
> *не суть важно*
>
> *важно то, что если объект являеся callable, его можно вызвать через круглые скобки*
>
> *callable1(1, 2) ----->*
>
> *callable2(1, 2) ----->*
>
> *ответ в обоих случаях - 3*
>
> *на бытовом уровне -- callable это функция*
>
> *ну или класс, который под нее мимикрирует*
>
> *дальше*
>
> *чем хороша функция*
>
> *ее можно вызвать и она может глючить*
>
> *далее*
>
> *если для кого-то октровение -- то в NOC дофига функций*
>
> *он вообще весь из них состоит*
>
> *поехали дальше*
>
> *функцию можно вызвать по имени*
>
> *на самом деле -- по id*
>
> *если я скажу python'y callable1, он мне даст ссылку на функцию*
>
> *это как указатель на функцию в C*
>
> *когда известно, какие функции нужно вызывать, их можно просто прописать в коде*
>
> *получим так любимый всеми хардкод*
>
> *теперь представьте, что мы знаем, что в каком-то месте должна быть обработка результата*
>
> *но понятия не имеем, какая*
>
> *это должен определить внедренец или админ*
>
> *нам нужно дать ему механизм вклинить в процесс свой функционал*

например, указав какую функцию вызвать

достаточно только определить, что на вход пойдет, и что она отдаст

итого у нас получается код, поведение которого можно менять не трогая сам код

мракетологи называют это плагинами и другими бесовскими словами

далее возникает вопрос -- как их указывать

как нам понять, что нужно вызвать, если у нас этого кода нет
тут вариантов масса -- от динамических библиотек до хитрой правки кода
но, чтобы было проще, я сделал функцию get_solution, почему solution, позже расскажу
она на входе принимает полный путь к callable
на выходе выдает сам callable
то есть
f = get_solution("noc.sa.<blabla>.mymodule.myfunc")
f(1, 2) ---> 3
нам достаточно сделать модуль mymodule.py и определить в нем функцию myfunc
в приницпе это ровно то же, что делается в pyrule
только прямее
то есть мы вот эти имена выносим в конфиги
далее
как с этим дальше жить
как только вы начинаете хачить код, возникает вопрос, как его сопровождать
наиболее разумное -- собрать все свои локальные правки в одном месте отдельно от кода NOC и просто подключить их
например профили и скрипты, обработчики событий и прочее
получается один или несколько пакетов
в принципе, если пакет получился достаточно функциональным - его можно подарить кому-то еще
тут руки развязаны полностью
вот этот пакет называется solution
это не моя придумка, такие пакеты есть во многих OSS системах
и, порой, стоят в десятки раз дороже самой системы
не самая плохая практика

далее

вернемся к событиям

классификатору на вход поступило событие

он, обычно, реагирует согласно атрибутам event class'a

но настройки на все случаи жизни в event class'е не предусмотришь

и часто нужно смотреть и настройки объекта, и профиля и просто фазу луны

порой проще сказать -- у меня тут есть несколько полезных функций, если придет событие этого класса - прогони через них

в json эти функции указываются ровно тем же механизмом по полному имени

например -- логгирование логинов/логаутов в отдельной коллекции, ведение списка набранных комманд и прочее

solution'ы содержат API для того, чтобы вклиниться в эти настройки и довернуть еще свои обработчики

делать их просто

def handler(event):

....

и все

у нас на входе instance ActiveEvent со всеми его методами

и с ним можно делать что угодно

ORM сам по себе дает хороший API для манипулирования данными

аналогичные петли есть во многих местах

скажем можно навесить свой алгоритм классификации событий

их даже сейчас из коробки два идет -- тупой и акселерированный

дальше, что такое job

пока в разрезе микросервисов, там прямее

{ "_id" : ObjectId("563f09d89d571742cace1091"), "jcls" :

"noc.services.discovery.jobs.box.job.BoxDiscoveryJob", "key" : 9, "ts" :

ISODate("2015-11-13T14:07:44.995Z"), "s" : "W", "data" : { }, "runs" : 67, "o" :

0.29027818407475636, "f" : 0, "ldur" : 1.6495530605316162, "st" :

ISODate("2015-11-13T11:07:46.849Z"), "ls" : "S" }

это примерно такая структурка в монге

jcls -- как раз такое вот имя callable

ему на вход дают параметры job'a, по завершению он сам решает, когда запуститься в следующий раз

отличие микросервисов -- в job можно пихнуть любую функцию

там есть специальный job под названием call

то есть механизм получается тот же самый

но у вызова появляется расписание

например, при сохранении МО я могу не пересчитывать все в контексте вызова save, а отдать на отработку совсем в другой процесс