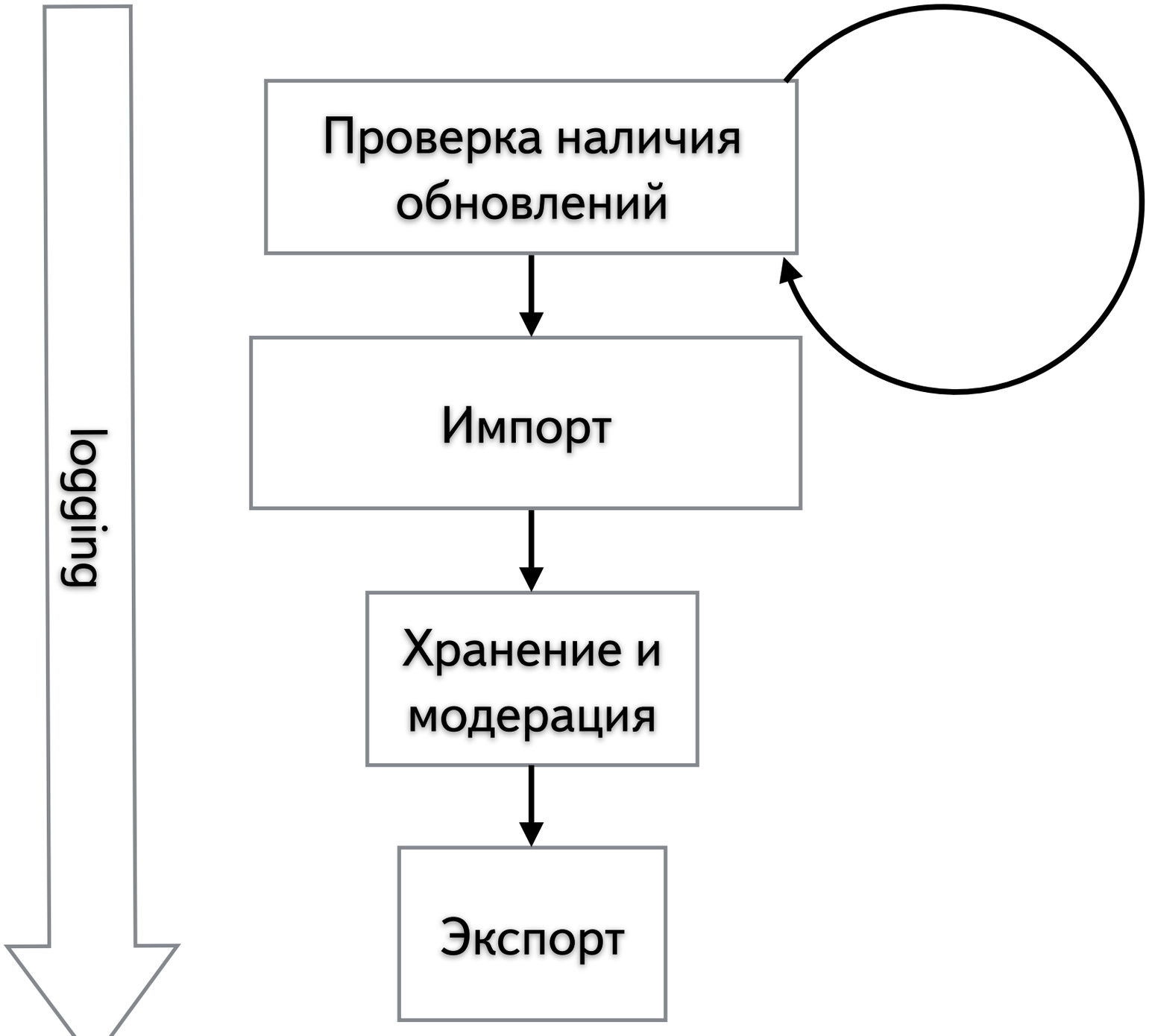
Яндекс

Использование распределенной очереди задач для обработки данных в бэкенде Яндекс Спорта

Александр Южаков

20 сентября 2014

Агрегация данных



Задачи решаемые на бэкенде

- Импорт и обработка данных
- Валидация импорта
- Интернационализация
- Хранение данных, модерация

Задачи решаемые на бэкенде

- Подготовка данных (экспорт) для API и фронта
- Полное логирование процессов импорта/экспорта
- Независимость модулей различных видов спорта

Задачи решаемые на бэкенде

- Высокая скорость работы
- Балансировка нагрузки
- Отказоустойчивость

Особенности спортивных проектов

- Данных не очень много
- Строгие требования к свежести данных
- Множество источников, различные форматы
- Схожие сущности в различных видах спорта будут иметь разные схемы данных

Особенности спортивных проектов

- Большие пиковые нагрузки, при относительно низких средних нагрузках
- Всегда есть задержки в данных относительно реальности
- Дедлайн не перенести

спортсмены

...

расписание

...

результаты

составы команд

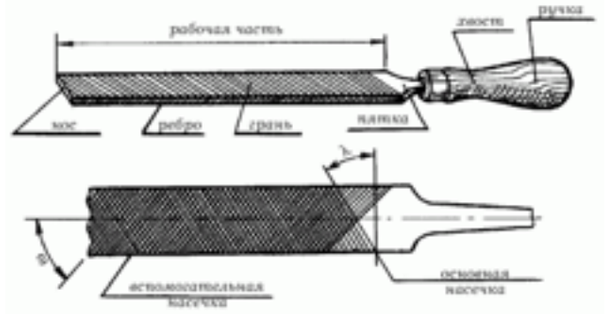
...

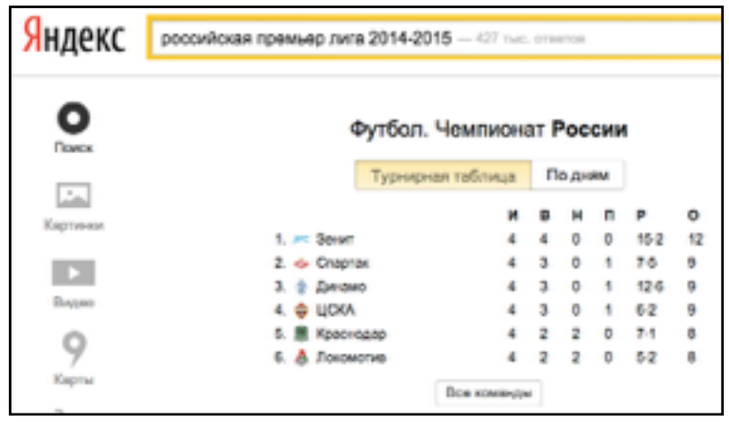
расписание ТВ

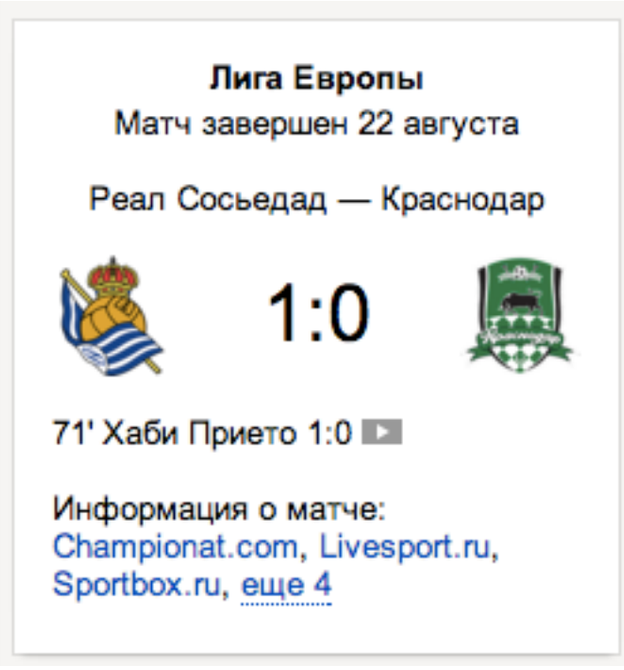
...

онлайн трансляции

видео лучших моментов и голов







Инструментарий

Celery 3.1.x

- Python 2.x
- Django
- MySQL
- MongoDB

Celery

```python
# предварительно необходимо сконфигурировать Celery
# и запустить воркеры

from celery import task

@task
def my_super_task(*args, **kwargs):
    do_some_magic_here()
```

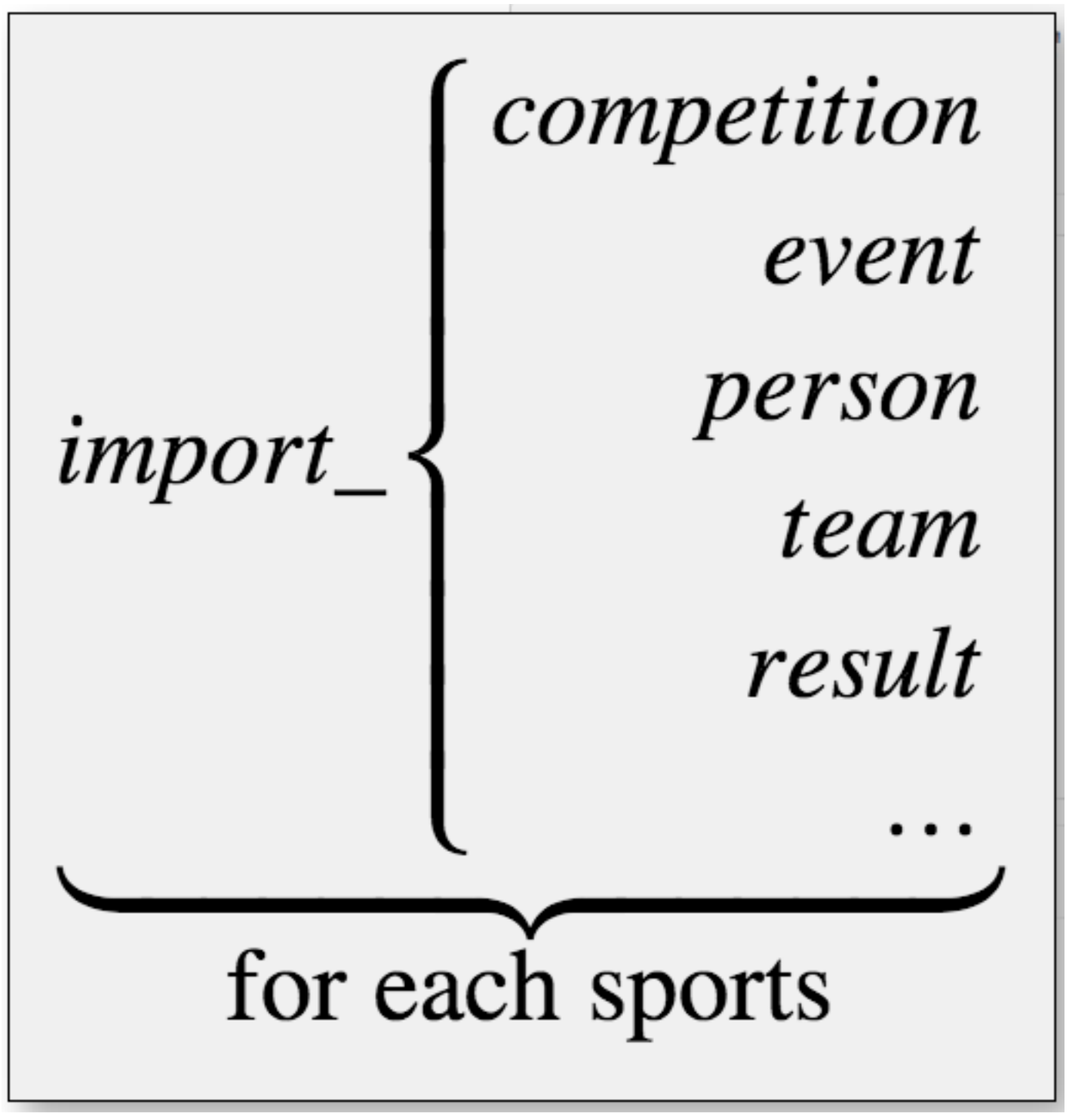
```python
from celery import Task


class MyMegaTask(Task):
    def __init__(*args, **kwrags):
        initialization(*args, **kwargs)

    def run(*task_args, **task_kwargs):
        task_job_here(*task_args, **task_kwargs)
```

Задачи импорта в виде функций

@task

def 

(...):

Задачи импорта в виде функций

```python
@task
def import_smth(source):
    raw_data = get_data_from_source(source)
    data = parse_and_validate_data(raw_data)
    import_results = import_data(data)
    do_smth_after_import(import_results)
```

Плюс

+ просто и очевидно

Задачи импорта в виде функций

Минусы:

- многократное повторение одного и того же кода
- нет возможности использовать уже готовые функции импорта/экспорта с небольшими изменениями
- трудно отслеживать движение задачи
- сложно обрабатывать исключения

К чему мы пришли...

Задачи как классы

```python
class BaseTask(Task):
    abstract = True

    @classmethod
    def task(cls):
        return cls.app.tasks[cls.name]

    @property
    def type(self):
        ...

    def on_success(self, retval, task_id, args, kwargs):
        ...

    def on_retry(self, exc, task_id, args, kwargs, einfo):
        ...

    def on_failure(self, exc, task_id, args, kwargs, einfo):
        ...

    def delay(self, *args, **kwargs):
        ...
```

Базовый класс для импорта

```python
class Importer(BaseTask):
    abstract = True
    ...  # class fields

    @classmethod
    def import_data(cls, url, params=None, *args, **kwargs):
        ...

    @classmethod
    def prepare_cache(cls, data, params=None, **kwargs):
        ...

    @classmethod
    def import_chunk(cls, chunk, cache, params=None, **kwargs):
        raise NotImplementedError

    def run(self, url, params, *args, **kwargs):
        key = md5(url).hexdigest()
        with MongoLock('import_locks', key) as lock:
            self.import_data(url, params, *args, **kwargs)
```

```python
@classmethod
def import_data(cls, url, params=None, *args, **kwargs):
    try:
        data = cls.prepare_and_validate(url, params=params, **kwargs)
        cache = cls.prepare_cache(data, params, **kwargs)
    except ValidationError, e:
        ...  # отбрасываем данные как плохие, с обработкой ошибки

    cls.before_import(data, cache, **kwargs)
    changed_objects = []
    for chunk in data:
        try:
            inst, changed = cls.import_chunk(
                chunk, cache, params=params, **kwargs
            )
            if cls.multilang:
                cls.import_translations(chunk, cache, **kwargs)
            if changed:
                changed_objects.append(inst.pk)
        except SkipImport as e:
            ...  # отбрасываем только часть данных

    cls.after_import(data, cache, **kwargs)
    if changed_objects:
        ...  # автоэкспорт объектов которые изменились
```

Пример импорта команд в футболе

```python
# def prepare_cache(...):

@classmethod
def import_chunk(cls, data, cache, params=None, **kwargs):
    match, created = models.Match.objects.get_or_create(
        outer_id=data['n_MatchID'],
    )

    data.update({
      'team1':                 cache['teams'].get(data['n_HomeTeamID']),
      'team2':                 cache['teams'].get(data['n_AwayTeamID']),
      'n_Result1':             data['n_HomeGoals'],
      'n_Result2':             data['n_AwayGoals'],
      'score':                 data['c_Score'],
      'score_suffix':          data['c_ScoreSuffix'],
      'team1_goals_shootout': data['n_HomeGoalsShootout'],
      'team2_goals_shootout': data['n_AwayGoalsShootout']
    })

    return match, update_instance(match, data, created, **kwargs)
```

Плюсы?

+ Минимум кода

+ Фокус на импорте данных без явного взаимодействия с подсистемами

+ Обработка ошибок через генерацию стандартных исключений (SkipImport, ValidationError)

Плюсы?

+ Декларативность
+ Стандартизация логирования
+ Предварительное кэширование для оптимизации работы с БД
+ Многоуровневая абстракция

Минусы?

– Стал выше порог вхождения для программиста

Еще много чего интересного...

- Классы экспортеры
- Классы импорта/экспорта переводов из Танкера
- Удобное централизованное логирование

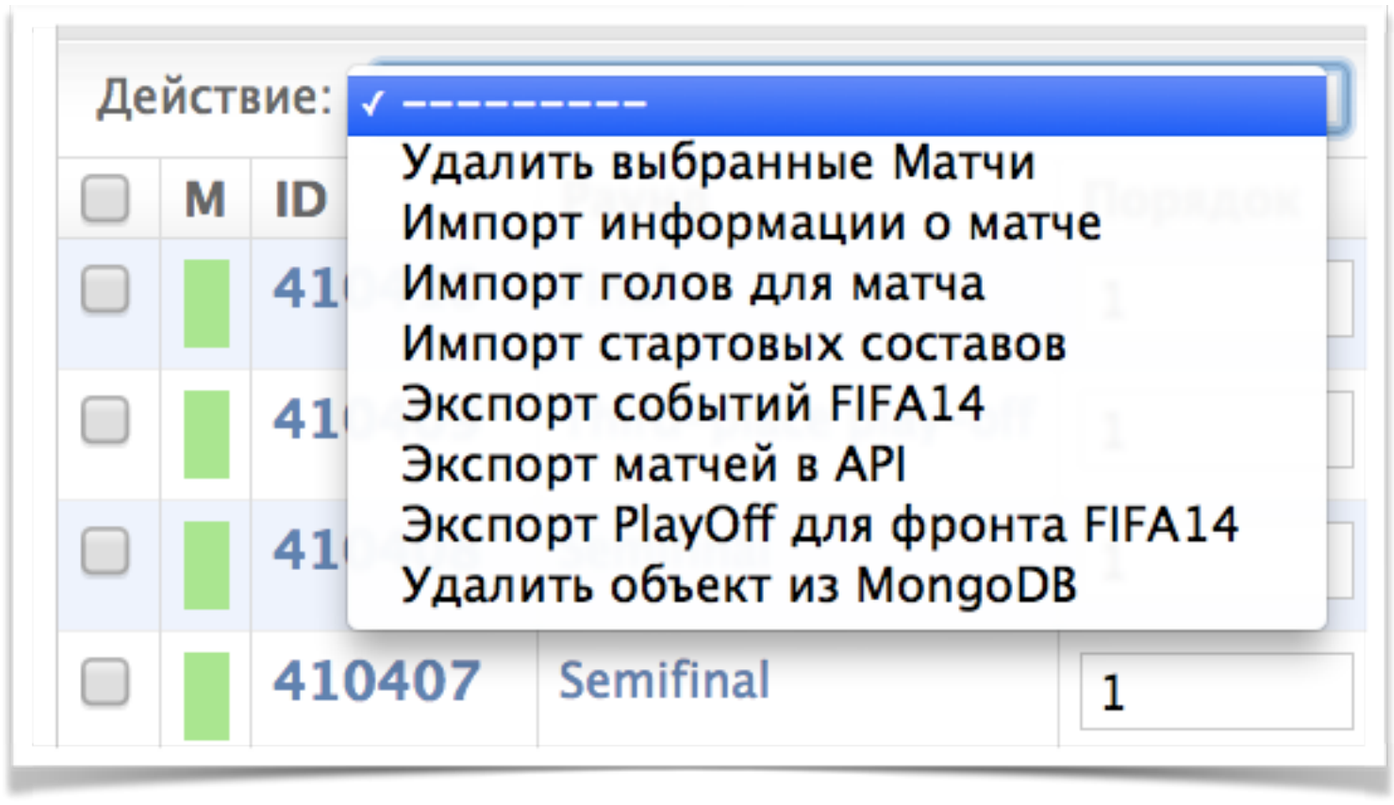
Логи

eb41e6a8a16b	football.MatchInfoImporter	Finished
eb41e6a8a16b	football.MatchInfoImporter	New task (PlayoffExporter) enqueued: e945b8b66d43
eb41e6a8a16b	football.MatchInfoImporter	New task (MatchAPIExporter) enqueued: f7fad79d6683
eb41e6a8a16b	football.MatchInfoImporter	New task (EventsExporter) enqueued: 712de80d1e82
eb41e6a8a16b	football.MatchInfoImporter	Match with id = 423069 was changed Details here.
eb41e6a8a16b	football.MatchInfoImporter	In progress.
eb41e6a8a16b	football.MatchInfoImporter	Import url: http://yandex.api.infostradasports.com/svc/Football.sv matchid=1905292&languagecode=2
eb41e6a8a16b	football.MatchInfoImporter	Task enqueued.

Еще много чего интересного...

- Импорт переводов (если данные на разных языках)
  import_translations(data, cache, *args, **kwargs)
- Интеграция в Django-админку
- Независимость модулей системы
- Класс Resolver для автоимпорта по URL

Админка


Действие:
✓ ---------
Удалить выбранные Матчи
Импорт информации о матче
Импорт голов для матча
Импорт стартовых составов
Экспорт событий FIFA14
Экспорт матчей в API
Экспорт PlayOff для фронта FIFA14
Удалить объект из MongoDB
M
ID
410407
Semifinal
1

Спасибо за внимание!