

Знакомимся с YAML

Сергей Супрунов

Как программисты-разработчики, так и системные администраторы не понаслышке знают, что такое рутина. Учитывая, что рутинные операции должен выполнять всё-таки компьютер, посмотрим, чем нам поможет язык YAML в таком деле, как обработка конфигурационных файлов для разрабатываемого приложения.

Чтобы приложение было гибким и настраиваемым, оно снабжается конфигурационным файлом (а то и несколькими). Очевидно, что формат этого файла должен соответствовать определённым соглашениям, то есть можно сказать, что он пишется на некотором языке разметки. И при выборе такого языка нужно учитывать особую роль конфигов – ведь они создаются людьми для программ. С одной стороны, такой файл должен легко читаться и редактироваться человеком (не всегда имеющим диплом с отличием по специальности «Программист»), а с другой – легко «разбираться» программой. Причём второе требование связано не столько с ресурсоёмкостью задачи «разбора», сколько с простотой разработки соответствующей подпрограммы. Особенно актуально это становится для скриптовых языков (Perl, Python, Ruby и т. п.), основное назначение которых – быстрая разработка

чего-нибудь небольшого. Согласитесь, было бы нелепо для скрипта в 50-100 строк писать аналогичного или даже большего размера «парсер» для разовой (в контексте работы скрипта) задачи анализа конфигурации.

Естественно, как только в процессе работы возникает подобная проблема, то в голову сразу приходит бессмертная фраза: «Всё украдено до нас». В том смысле, что если разработчик сталкивается с некоторой типовой задачей, то с высокой долей вероятности можно считать, что данная задача уже кем-то решена. Поэтому и в части анализа конфигурационных файлов проблема обычно сводится к поиску наиболее удобного инструмента среди уже существующих.

Краткий обзор «типичных» форматов

Наиболее простым форматом конфигурационного файла можно считать

«параметр-разделитель-значение» (чаще всего в качестве разделителя выступает любое количество пробелов и/или символов табуляции, но могут быть и другие, скажем, знак равенства или двоеточие; сокращённо будем называть этот формат «П-Р-З»):

```
node_ip      10.0.0.51
node_user    arc
node_password xidughei
email_list   arc,hunter,engineer
sms_list     hunter12@sms.mob.ru
```

Его достоинства: простота написания и восприятия человеком и простой разбор (функцией split по регулярному выражению, которая имеется практически во всех современных скриптовых языках). Но есть и недостатки: «линейность», т.е. невозможность создавать иерархию параметров, и «нетипизированность», т.е. сложности с передачей параметров, тип которых не укладывается в понятие «простого» (строка, число).

Для борьбы с первым недостатком используются различные методы. Например, разделение файла на так называемые «секции», когда секция вводится специальным синтаксисом (скажем, «[Server]»), и все нижеследующие параметры вплоть до маркера другой секции рассматриваются как относящиеся к ней (вы тоже вспомнили про win.ini?). Второй недостаток вынуждает вводить дополнительные соглашения, и при разборе конфигурации учитывать их (в примере выше придётся особо оговорить, что список параметров, разделённых запятыми, является массивом, а при считывании конфигурационного файла потребуются как минимум ещё один вызов split).

Другой подход к описанию конфигурации – XML-подобный синтаксис, набирающий в последнее время всё большую популярность (сам не видел, но рассказывают, что в MacOS даже настройки операционной системы в этом формате задаются). Вышеприведённая конфигурация на этом языке могла бы выглядеть как-то так:

```
<node>
  <ip>10.0.0.51</ip>
  <user>arc</user>
  <password>xidighei</password>
</node>
<list type='email'>
  <recipient>arc</recipient>
  <recipient>hunter</recipient>
  <recipient>engineer</recipient>
</list>
<list type='sms'>
  <recipient>hunter12@sms.mob.ru
  </recipient>
</list>
```

Здесь налицо возможность строить иерархию любой сложности. Дополнительные атрибуты также заметно расширяют возможности этого формата. Собственно, было бы странно ожидать меньшего от языка, претендующего на роль универсального средства обмена информацией. Только вот анализ такого файла простым никак не назовёшь (хотя для тех, кто «на ты» с XPath, XSLT и прочими мудрёными аббревиатурами, может быть, и ничего сложного). Да и редактировать вручную такого монстра не слишком удобно – в глазах рябит от тегов разметки.

Нечто среднее между этими двумя подходами мы видим в конфигурации Apache, ProFTPD и ряде других приложений – «тегами» вводится нужная

О примере

По ходу статьи часто приводится пример некоторого конфигурационного файла. Чтобы придать ему немного конкретики, будем считать, что принадлежит он скрипту, задача которого – периодически контроли-

ровать с помощью FTP наличие некоторого файла на сервере (node_ip), и в случае неудачи отправлять уведомления на электронную почту и в виде SMS пользователям согласно заданным спискам.

```
<Node>
  ip      10.0.0.51
  user    arc
  password xidighei
</Node>
<Lists>
  email    arc,hunter,engineer
  sms      hunter12@sms.mob.ru
</Lists>
```

В различных вариантах подобный подход можно найти и в конфигурации других продуктов. Читать и редактировать – удобнее, насчёт удобства программного анализа не вполне уверен. Да и в любом случае это уже похоже на «частное» решение.

Поскольку при разработке скриптов, как правило, хочется чего-то совсем простого, то не последнее место по популярности занимает «программистский» подход: в качестве файла конфигурации используется подключаемый скрипт на «рабочем» языке, сразу при своём выполнении создающий нужный набор «конфигурационных переменных» (пример на языке Python):

```
node = {'ip': '10.0.0.51',
        'user': 'arc',
        'password': 'xidighei'}
email_list = ['arc', 'hunter', 'engineer']
sms_list = ['hunter12@sms.mob.ru']
```

Типичный пример такого подхода – большинство конфигов *BSD или Linux, представляющих собой скрипты на языке командной оболочки (sh, реже csh или bash). Очевидно, что здесь совершенно не стоят проблемы разбора конфигурации, структурных ограничений или невозможности задать нужный тип. Но при этом от пользователя уже начинают требоваться хотя бы базовые познания в используемом языке программирования и немалая аккуратность и внимательность. Да и разработчик скрипта, если он стремится создать достаточно надёжную программу, должен в своём коде учитывать возможность опечатки (скажем, забыли

заклЮчить текстовый параметр в апострофы) и соответствующим образом такие ситуации обрабатывать.

Теперь настало время перейти собственно к теме этой статьи – языку YAML, набирающему всё большую популярность и стремящемуся оставаться максимально простым как для человека, так и для программы.

YAML – «ещё один» или «совсем не»?

Разработанный Кларком Эвансом, язык YAML (в версии 1.0 эта аббревиатура «официально» расшифровывалась как «Yet Another Markup Language», начиная с 1.1 разработчики решили, что «YAML Ain't Markup Language») первоначально претендовал на роль полноценной замены XML, но затем акцент сместился в сторону максимально гибкого описания данных. Он позволяет наиболее полно отображать основные типы данных, используемых современными языками программирования, сохраняя при этом максимальную простоту как для человека, так и для программного анализа. Wikipedia называет YAML форматом сериализации данных (<http://ru.wikipedia.org/wiki/YAML>), что очень точно отражает его основную задачу. И действительно, YAML можно с успехом использовать для маршалинга (т.е. для представления данных в памяти в формат, пригодный для хранения и передачи, см. <http://ru.wikipedia.org/wiki/Маршалинг>) или постоянного хранения на диске тех или иных структур данных: скажем, в языке Python для сериализации традиционно используется модуль pickle, в Ruby часто используют модуль Marshal, аналогичные решения есть и в других языках. Но если возникает необходимость организовать обмен данными между скриптами, написанными на различных языках программирования, то уже требуется какой-нибудь универсальный язык сериализации. YAML может использоваться в этой роли почти без ограничений.

Тем не менее основная цель данной статьи – рассмотреть YAML как язык конфигурации. В этом качестве он успешно используется в таких фреймворках, как Ruby on Rails, Symfony, Spring. Что же он из себя представляет?

Наш пример на YAML может выглядеть так:

```
node:
  ip: 10.0.0.51
  user: arc
  password: xidighei

email_list:
  - arc
  - hunter
  - engineer

sms_list:
  - hunter12@sms.mob.ru
```

Как видите, разработчикам YAML удалось сохранить лёгкость и «читабельность» простейшего формата с разделителями, сделав его иерархическим за счёт использования вложенных параметров. Да и возможности работы с типами данных здесь заметно расширены. Например, то, что вы видите выше, это самый настоящий хэш («словарь» в терминологии Python), один элемент которого представляет собой вложенный хэш, а ещё два – массивы («списки» в Python).

Вот, например, как это «развернётся» в Ruby (вывод слегка отформатирован, для удобства восприятия):

```
$ irb
irb(main):001:0> require 'yaml'
irb(main):002:0> config = YAML::load(open('config.yml'))
=> {"node"=>
  {"user"=>"arc",
   "ip"=>"10.0.0.51",
   "password"=>"xidighei"},
  "sms_list"=>["hunter12@sms.mob.ru"],
  "email_list"=>["arc", "hunter", "engineer"]}
irb(main):003:0> puts config['node']['ip']
10.0.0.51
```

Как видите, хэш на YAML задаётся парами «ключ: значение», причём ни для ключа, ни для значения практически никаких ограничений не существует – они могут содержать пробелы, быть многострочными, представлять собой вложенные элементы... Да-да, ключ хэша может быть сложной структурой данных! В этом случае он начинается с символа «?». Правда, далеко не все языки программирования допускают подобные вольности. В частности, пример, показанный ниже, нормально отрабатывается в Ruby, а, например, в Python уже вызывает ошибку.

```
$ cat hash.yml
? - 1
  - 2
  - 3: three
:
  some digits
$ irb
irb(main):001:0> require 'yaml'
irb(main):002:0> config = YAML::load(open('slohash.yml'))
=> {[1, 2, {3=>"three"}]}=>"some digits"
irb(main):003:0> config.keys[0]
=> [1, 2, {3=>"three"}]
irb(main):004:0> config[c.keys[0]]
=> "some digits"
```

Ещё одна возможность – использовать в качестве ключа хэша различные символы (если ваш язык программирования от этого не впадёт в ступор):

```
+: plus
:: colon
```

После обработки, скажем, в Python получим такой словарь: «{'+': 'plus', '::': 'colon'}».

Кроме показанного выше синтаксиса, для массивов и хэшей существует и «линейная» нотация, близкая по виду к языку Python (кстати, отступы в YAML тоже играют не последнюю роль, так что любителям Python он явно придётся по вкусу):

```
# Массив (как элемент хэша)
Moderators: [Site Admin, Dr.Moder, Polizei]

# Хэш (как элемент хэша)
Location: {host: localhost, port: 5432}
```

Самой собой, что с обычными строками и числами проблем тоже не возникает. Параметры, заданные «в виде числа», включая восьмеричное и шестнадцатеричное представление, а также научную нотацию, автоматически преобразуются в числовой формат. Аналогично, автоматическое преобразование предусмотрено для дат. Скажем, строку '2008-01-05' модуль PyYAML языка Python преобразует в выражение «datetime.date(2008, 1, 5)». В качестве примера рассмотрим обработку такого файла:

```
Date1: 2008-01-01
Date2: 01.02.2008
Time: 12:45
StringDate: "2008-01-01"
```

Если этот файл называть «config.yml», то работать с ним можно следующим образом:

```
$ python
>>> import yaml, datetime
>>> config = yaml.load(open('config.yml'))
>>> config

{'Date1': datetime.date(2008, 1, 1), 'Date2': '01.02.2008',
 'Time': 765, 'StringDate': '2008-01-01'}

>>> config['Date1'].year

2008
```

Заметили, что как дата представляется только строка в формате ISO (YYYY-MM-DD)? Если говорить точнее, то спецификация поддерживает два формата даты и времени:

- Date1: 2008-08-03t11:00:00
- Date2: 2008-08-03 11:00:00

При необходимости можно задать и временную зону. Однако различные «европейские» форматы (вроде «01.01.2008») обработчиками YAML именно как даты не распознаются. Также обратите внимание на то, что строка «Time: 12:45» была преобразована в «Time: 765». Дело в том, что YAML значения вида «XX:YY:ZZ» распознаёт как шестидесятеричные числа. То есть запись «12:45» была рассчитана как 12*60 + 45. Фактически, подразумевая под этой записью «12 часов 45 минут», мы получили число минут с начала дня.

Также следует учесть возможности YAML по явному и неявному преобразованию типов данных: если вам нужно передать значение параметра именно как строку, просто за-

ключите его в апострофы или кавычки (в кавычках будет выполняться замена escape-последовательностей, строка в апострофах используется без каких-либо преобразований). Есть и более строгий способ указать тип данных:

```
PortAsString: !!str 80
```

Ещё одна возможность, особенно полезная для конфигурационных файлов, – поддержка своего рода «указателей», когда вы можете присвоить некоторому значению имя (вводится символом «&»), а затем «разыменовывать» его (символом «*») в других местах файла:

```
Email list:
- arc
- &h hunter
- engineer
SMS list:
- *h # заменится на hunter
```

Правда, использовать такую «переменную» как часть другого параметра (скажем, задав значение в виде «*h@sms.mob.ru») не получится.

Наконец, строки. Как вы уже поняли, строкой будет всё, что заключено в кавычки или апострофы, а также любые значения параметров, не опознанные как относящиеся к другому типу данных. Но YAML предоставляет и ряд дополнительных возможностей, например, поддержку «многострочных» строк (обращайте внимание на символ после двоеточия):

```
simple:
  one
  two
  three

continued: >
  one
  two
  three

pre: |
  one
  two
  three
```

Результат:

```
>>> print config['simple']
one two three

>>> print config['continued']
one two three

>>> print config['pre']
one
two
three
```

Вариант «simple» отличается от «continued» тем, что в последнем случае сохраняется завершающий перенос строки. Кроме того, промежуточные переводы строки после символа «>» будут заменяться на пробелы только для групп строк, имеющих одинаковый отступ. Если же


data center

Для тех, кто знает цену информации!

Data Center Forum – 2008

Форум по центрам обработки данных
30 сентября 2008
ВЦ «ИнфоПространство»

Основные темы Data Center Forum – 2008:

- системы управления данными и их хранения
- безопасность ЦОД
- преимущества и недостатки удаленных ЦОД
- непрерывность работы ЦОД
- типичные ошибки при построении ЦОД
- электронный документооборот в ЦОД
- возможности использования ЦОД в различных отраслях экономики (розничная торговля, телекоммуникации, банковский и финансовый сектор, промышленность)

Ключевые доклады Data Centre Forum – 2008:

- **Джил Экхос**, генеральный директор и член правления Ассоциации менеджеров информационных центров (AFCOM)
- **Леонард Экхос**, основатель и ex-президент AFCOM

Организатор



Приглашаем ИТ-директоров, специалистов по обработке данных, руководителей технических отделов!

Контактная информация:

dcw2008@fort-ross.ru

+7 812 334-16-86

www.datacenter-forum.ru

Зарегистрируйтесь сегодня – при ранней регистрации предусмотрена гибкая система скидок!

Подробнее о спонсорских возможностях на www.datacenter-forum.ru

перед строкой стоит символ «|», то все переводы строки сохраняются в неприкосновенности.

Также существует возможность хранить двоичные строки в формате Base64:

```
png: !!binary |
iVBORw0KGgoAAAANSUuEugAAABUAAAAVCATAAAmdTLBAAAAAXNSR0IA
rs4c6QAAAAARnQU1BAACxjwv8YQUAAAgY0hSTQAAeiYAAICEAAD6AAAA
gOgAAHUwAADqYAAAOpgAABdwnLpRPAAAAh5JREFUOE+1VEsoRFEYPooW
xORRk5SFBQsLNRtrS1ZKKXaSEFeyQx5K6Q0iPFME6XIKYzjUKTEqMR
ZiFIMyk1Hum614zju3PvXGfunZFyOp05c+75vv///seJoZSS/wzgI4+
HK+q2051ucblzRbtFtB8Cp7NCE8ZrZ+rjkP7CSrD6evSfzSn+vW7t5XD
8i5cfLbofT11EutF2LSdEPdMEj+YT723LAUd9zg5U/r+phrJEm3ukbe
utM+TKYUihBe4D/MBZgc2dMULBapZ2ZCjsMyfixSKFzJesFWd2uJWCXij
0YhVAZgW9aBjKcFKCQ4gvg7FyIkAbR4HNB1GUor8lght1NJgcMRGSs
1jtXZNKR8eUluZgsfslBBHOWjPc3Ji6dheGr07JhExMbWib/lv0EUVb4
Jh198RiQcIH6WOWbZF/CQzP2rHKW4qlfTz1OQp/ubZkFV6JX+tkGiiw
qozjbwqPEbU/Odfqhy/wf+xIzA47Uad04UqMPyKBePwSf4RAFT9Ugb8h
AdpF/Ne2+XJWDoFUP2z+wSuVAKvCsUb4mZJQ/t99nCKTfa4VGfEEexp970+
F8C19ftx199kf89VyyNXC1F9wyy+UuZz/EhmN62CdcMMFUK7G4wgU8CK
aENQYWhsNTt99mv5TDtx2xMIzoUMjS0WJdXEX3EzGo7vxtKiekAjjv0h/
PA07qGuUBtYPSyFljKFate/PNwvDAKJ/owerAAAAElFTkSuQmCC
```

Теперь можно делать с этим изображением всё, что требуется (в примере ниже я сохраняю его в файл):

```
>>> c = yaml.load(open('r.yml'))
>>> open('r.png', 'w').write(c['png'])
```

Кроме рассмотренных выше, язык YAML умеет работать и с логическими типами данных («true» и «yes» распознаются как «истина», «false» и «no» — как «ложь»), поддерживаются специальные значения: «null» и «~» как «None», «.NaN» в значении «nan» (не число); «.inf» и «-.inf» — соответственно «плюс бесконечность» и «минус бесконечность».

Ещё следует знать о последовательности «----» — она является разделителем документов в потоке. Не все модули, работающие с YAML, способны считывать несколько документов из одного файла. Скажем, в Python метод load() модуля PyYAML считывает только первый (для работы со всеми документами следует использовать метод load_all(), возвращающий генератор, из которого можно последовательно, методом next(), выбрать все документы потока). Модуль 'yaml' в Ruby действует аналогично, за тем исключением, что если символами «----» разделять простые строки, то модуль включает их в состав строки, считая всё одним документом. Модуль Perl — YAML::Tiny — успешно работает с несколькими документами:

```
$ cat many.yml

--- # Server
host: localhost
port: 80
--- # Client
user: root
homedir: /var/client/home

$ cat perl.pl
#!/usr/local/bin/perl

use YAML::Tiny;

$config = YAML::Tiny->read('many.yml');
$server = $config->[0];
$client = $config->[1];

print "Connect to $server->{host}:$server->{port}\n";
print " as $client->{user} from $client->{homedir}\n";

$ ./perl.pl
Connect to localhost:80
as root from /var/client/home
```

Здесь many.yml — файл с двумя документами. Символом «#» начинаются комментарии.


Трудности перевода

Есть у YAML и недостатки. Один из самых существенных — проблемы с кодировками, отличными от Unicode. Спецификация предусматривает поддержку лишь UTF-8, UTF-16 LE и UTF-16 BE. Как результат — работа с другими кодировками полностью зависит от используемого модуля. Так, модуль PyYAML и во FreeBSD, где используется KOI8-R, и в Windows XP (CP1251) упорно не желал обрабатывать файлы, содержащие хотя бы один кириллический символ независимо от его месторасположения. С модулем 'yaml' в Ruby нигде проблем не возникло. В Ubuntu, где по умолчанию используется UTF8, все протестированные модули показали безупречную работу с кириллицей.

Таким образом, появился ещё один повод всюду, где только можно, переходить на Unicode.

Заключение

Как видите, YAML — довольно простой в освоении язык сериализации. Наличие готовых модулей для многих языков программирования позволяет использовать его «прямо сейчас», да и разработка собственного парсера вряд ли будет

слишком сложной задачей. Читается конфигурация на YAML очень легко, и в то же время вы можете не ограничивать себя только строковыми значениями параметров, используя и массивы, и хэши, и логические значения... Если вы заинтересовались данным языком — обязательно просмотрите его полную спецификацию. Наверняка вы найдёте ещё много интересного, что осталось за рамками статьи. 

YAML и языки программирования

Для работы с YAML практически каждый язык программирования имеет готовый модуль. Возможности YAML как языка сериализации достаточно обширны, но если говорить о таком узком применении, как конфигурационные файлы, то в большинстве случаев использование YAML сводится к вызову одного метода (обычно load()). Ниже рассмотрены типовые примеры использования для различных языков программирования (предполагается, что поток содержит один документ).

Perl:

```
use YAML::Tiny;
$config = YAML::Tiny-> \
    read('config.yml')->[0];
$host = $config->{'host'};
```

Python:

```
import yaml
config = yaml.load(open('config.yml'))
host = config['host']
```

Ruby:

```
require 'yaml'
config = YAML::load(open('config.yml'))
host = config['host']
```

В последних строках показаны примеры использования одного из параметров, в предположении, что он задан как элемент хэша (словаря).

Разрабатываются также модули/библиотеки для других языков программирования: C, PHP, Java, JavaScript, Haskell. Ссылки можно найти на главной странице сайта проекта: <http://yaml.org>.

1. Официальный сайт языка — <http://yaml.org>.
2. Спецификация YAML 1.2 — <http://yaml.org/spec/1.2>.
3. YAML Cookbook — <http://yaml4r.sourceforge.net/cookbook>.