

title

Tracを改造して別のブランチを  
～迷走編～

Y.Nakayama

2008年1月9日(土) Shibuya.trac #1

自己紹介します



## ■ 自己紹介

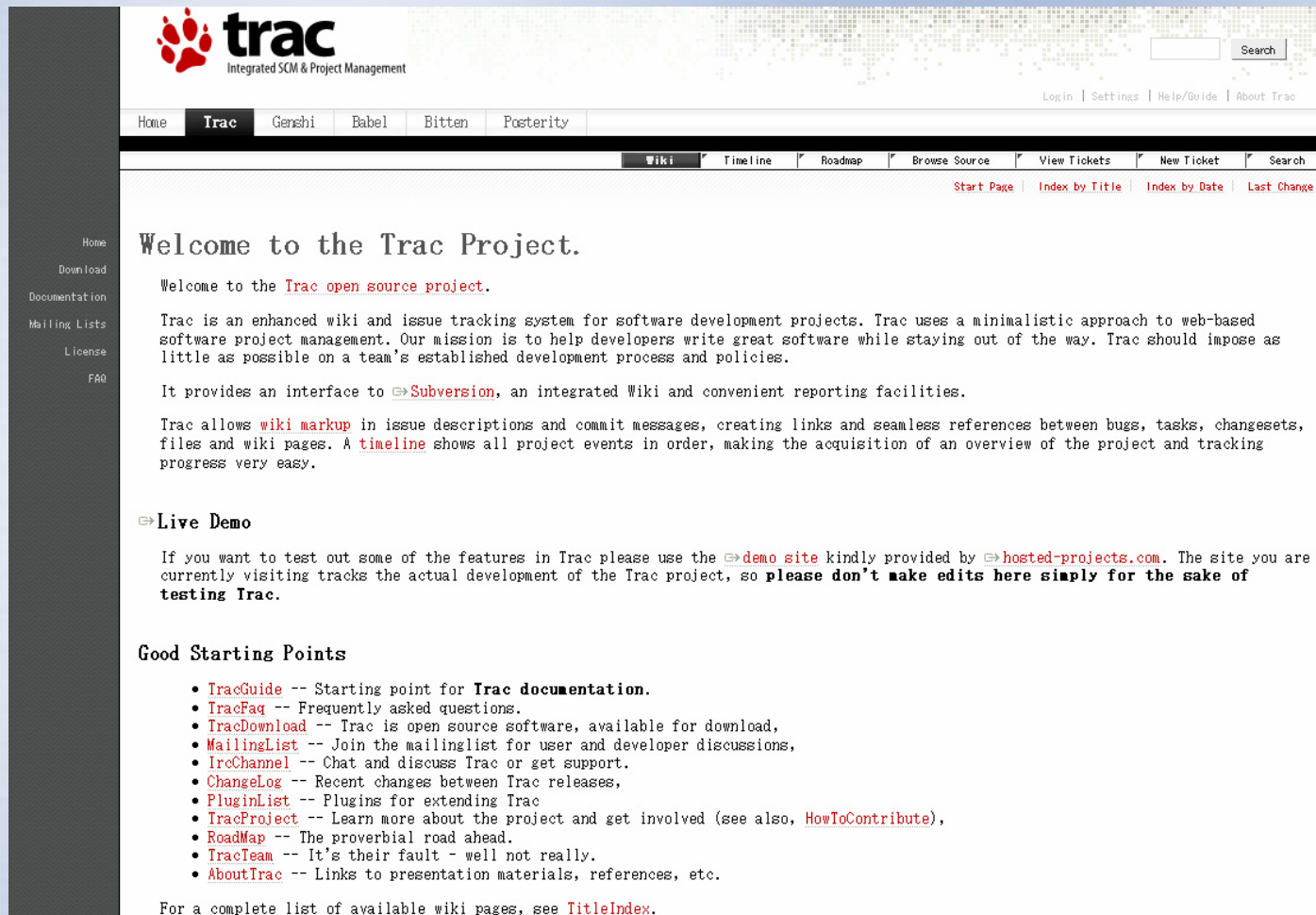
- 本業：ネット廃人←
- 副業：Sier

## ■ その他

- ※カットします



## ■ Trac



The screenshot shows the Trac project website. At the top left is the Trac logo, a red paw print, with the text "trac" and "Integrated SCM & Project Management" below it. To the right of the logo is a search box with a "Search" button. Below the logo is a navigation bar with links for "Home", "Trac", "Genshi", "Babel", "Bitten", and "Posterity". Below the navigation bar is a secondary navigation bar with links for "Wiki", "Timeline", "Roadmap", "Browse Source", "View Tickets", "New Ticket", and "Search". Below the secondary navigation bar is a third navigation bar with links for "Start Page", "Index by Title", "Index by Date", and "Last Change".

Home  
Download  
Documentation  
Mailing Lists  
License  
FAQ

## Welcome to the Trac Project.

Welcome to the [Trac open source project](#).

Trac is an enhanced wiki and issue tracking system for software development projects. Trac uses a minimalistic approach to web-based software project management. Our mission is to help developers write great software while staying out of the way. Trac should impose as little as possible on a team's established development process and policies.

It provides an interface to [Subversion](#), an integrated Wiki and convenient reporting facilities.

Trac allows [wiki markup](#) in issue descriptions and commit messages, creating links and seamless references between bugs, tasks, changesets, files and wiki pages. A [timeline](#) shows all project events in order, making the acquisition of an overview of the project and tracking progress very easy.

### ⇒ Live Demo

If you want to test out some of the features in Trac please use the [demo site](#) kindly provided by [hosted-projects.com](#). The site you are currently visiting tracks the actual development of the Trac project, so **please don't make edits here simply for the sake of testing Trac.**

### Good Starting Points

- [TracGuide](#) -- Starting point for **Trac documentation**.
- [TracFaq](#) -- Frequently asked questions.
- [TracDownload](#) -- Trac is open source software, available for download.
- [MailingList](#) -- Join the mailinglist for user and developer discussions.
- [IrcChannel](#) -- Chat and discuss Trac or get support.
- [ChangeLog](#) -- Recent changes between Trac releases.
- [PluginList](#) -- Plugins for extending Trac
- [TracProject](#) -- Learn more about the project and get involved (see also, [HowToContribute](#)),
- [RoadMap](#) -- The proverbial road ahead.
- [TracTeam](#) -- It's their fault - well not really.
- [AboutTrac](#) -- Links to presentation materials, references, etc.

For a complete list of available wiki pages, see [TitleIndex](#).

■ いきなりですがDIS

- Tracのコードはいけてない

■ DB設計

- ID列無し
- SQLがあっちにもこっちにも分散

■ MVC

- 統一性が無い...、なぜmodel.pyがあるのに他でもCRUDする

■ たまにテクニックを駆使する(クロージャとか)

■ 開発生産性

- どこかいじると、たちまち全体がうごかなくなる

■ でも目標は高く掲げられているわけですよ

- 全社標準・・・
- バージョンアップ・・・

■ 拡張できるのかこれ

- プラグイン作りってレベルじゃねえぞ！

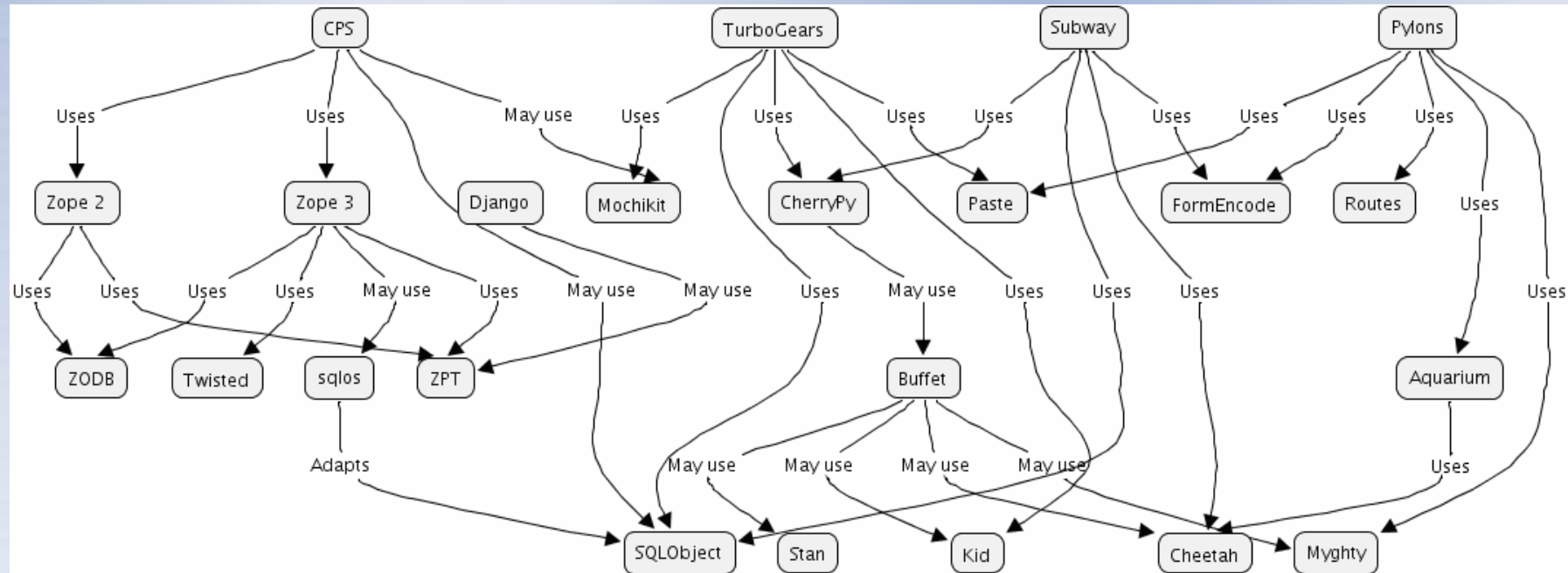


■ イチから作り直したくなるのがRubyistの性←

- 設計からきっちり作りこみたい
- 自分の理想どおりのものづくり



## ■ いきなりですがPythonのフレームワーク大杉



[http://blogs.nuxeo.com/sections/blogs/fermigier/2006\\_01\\_13\\_python-megaframeworks](http://blogs.nuxeo.com/sections/blogs/fermigier/2006_01_13_python-megaframeworks)

## ■ 現代のPythonにおける主要3フレームワーク

### ■ Django

- カンザスの新聞社のひとたちが記事を頻繁に更新して大量アクセスされても耐えうるよう、オールインワンなフレームワークをつくった。
- Python産みの親 Guido Van Rossum に褒められた
- 統一性高い、memcached内蔵、Ajax標準では無し

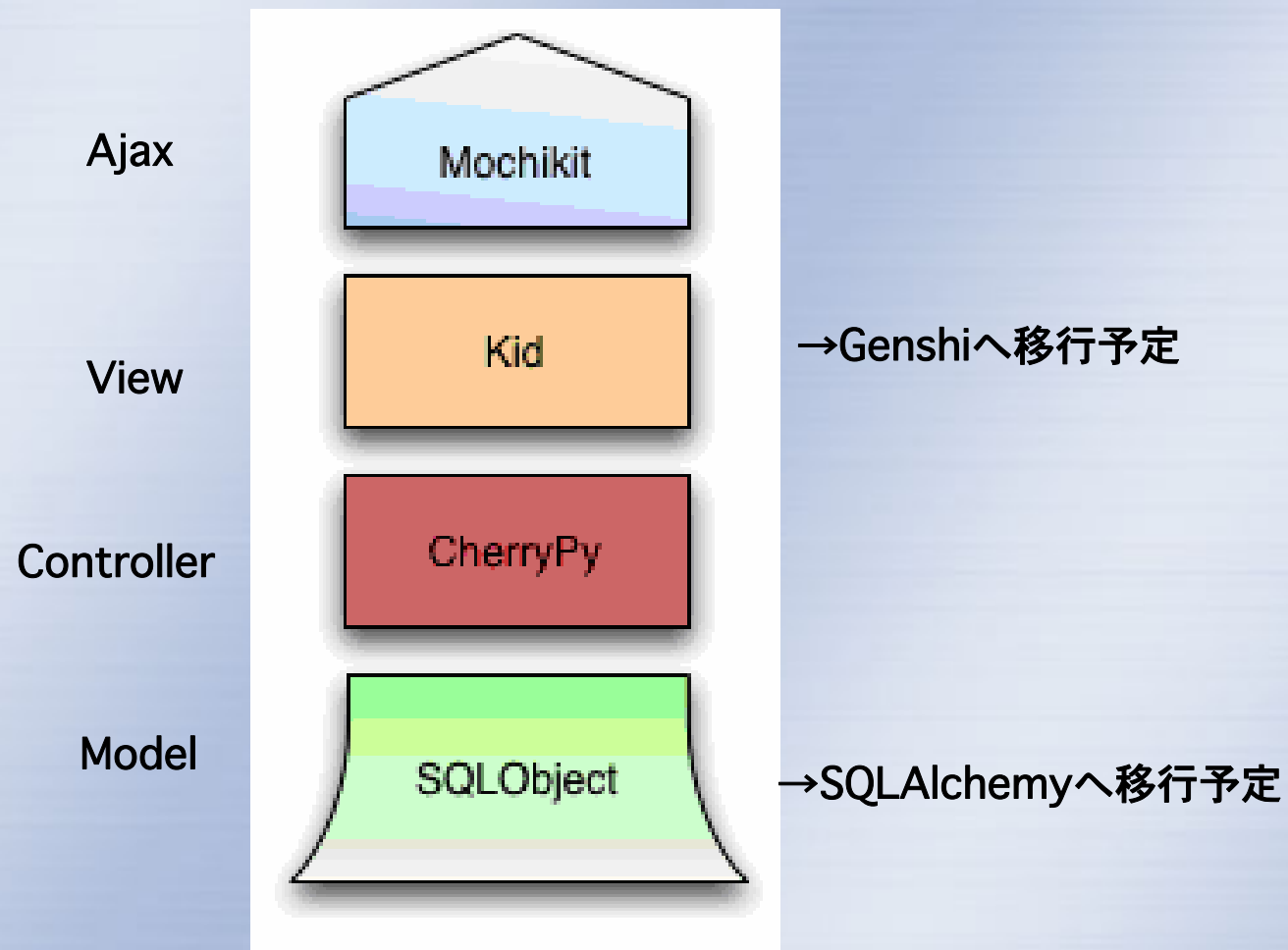
### ■ TurboGears

- RSSリーダーを作ろうとして既存フレームワークを組み合わせてみた。
- SQLAlchemy (M)、Kid (V)、CherryPy (C)、Mochikit (Ajax)
- 組み合わせ割と自由、覚えること大杉

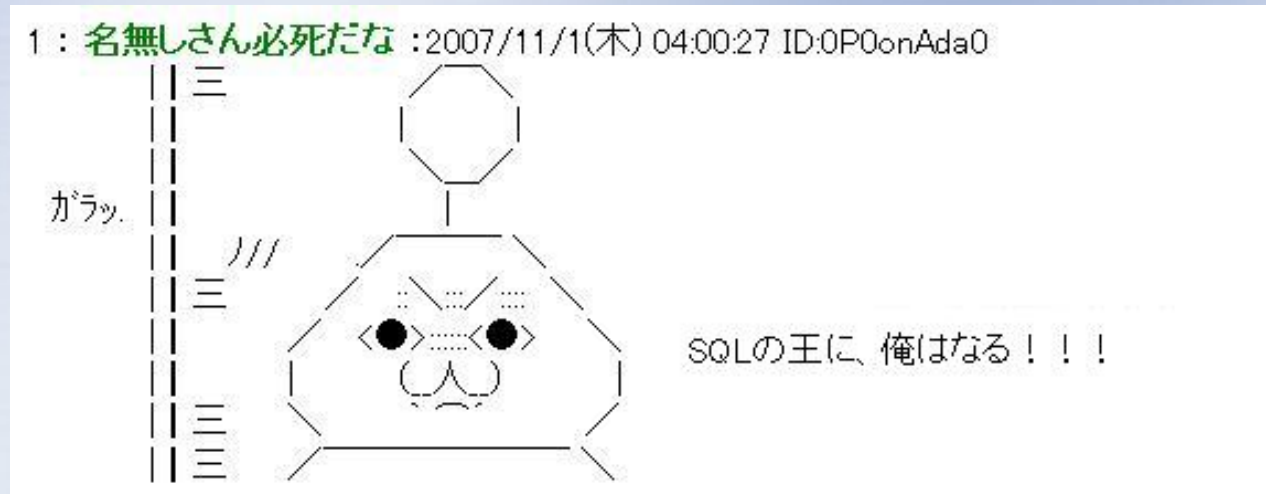
### ■ Pylons

- ぶっちゃけよく知らん
- TurboGears2が合流してPylons上に旧TGのAPIを作っていくとか

■ TurboGearsを構成するもの



## ■ さっそくSQLObjectを使おうじゃまいか



## ■ インストール

- `aptitude install postgresql python-psycopg2 python-sqlobject`

## ■ こんな感じで使う

### connect.py

```
class Database():
    def __init__(self):
        self.connection = 'postgres://' + ¥
        'postgres:xxxxxx@localhost/sampledb'
```

### model.py

```
from sqlalchemy import *
from connect import Database

db_conn = connectionForURI(Database().connection)
sqlhub.processConnection = db_conn

class Gakkyu(SQLObject):
    class sqlmeta:
        fromDatabase = True

class Seiseki(SQLObject):
    class sqlmeta:
        fromDatabase = True
```

■ SQLを書かずに、モデルとDBを1:1で対応させる(ActiveRecordパターン)

**insert/update**

```
yamada_s = Seiseki(name='YAMADA', kokugo=85, sugaku=95, eigo=45)
```

**select**

```
gakkyu_rows = Gakkyu.selectBy(gakunen='2')
```

**delete**

```
Seiseki.deleteBy(name=gakkyu_row.name)
```

## ■ Tracの場合

```
class Attachment(SQLObject):
    table = "attachment"
    type = StringCol()
    attach = StringCol()
    filename = StringCol()
    size = IntCol()
    time = IntCol()
    description = StringCol()
    author = StringCol()
    ipnr = StringCol()

class AuthCookie(SQLObject):
    table = "auth_cookie"
    cookie = StringCol()
    name = StringCol()
    ipnr = StringCol()
    time = IntCol()

class Component(SQLObject):
    name = StringCol()
    owner = StringCol()
    description = StringCol()
    class sqlmeta:
        idName = "name"
        idType = str
```

```
class Ticket(SQLObject):
    class sqlmeta:
        fromDatabase = True

class TicketChange(SQLObject):
    table = "ticket_change"
    ticket = IntCol()
    time = IntCol()
    author = StringCol()
    field = StringCol()
    oldvalue = StringCol()
    newvalue = StringCol()

class TicketCustom(SQLObject):
    table = "ticket_custom"
    ticket = IntCol()
    name = StringCol()
    value = StringCol()
```

## ■ 制約多すぎて面倒

```
try:
    p.write('%n[info] attachment created: id:' + ¥
            str(attachment_id_counter) + ', key:' + ¥
            ticket_dict[str(r['attach'])])
    ObjectCreator().attachment(attachment_obj, ¥
                                attachment_id_counter, ¥
                                r['type'], ticket_dict[str(r['attach'])], ¥
                                r['filename'], int(r['size']), int(r['time']), ¥
                                r['description'], r['author'], r['ipnr'])
except sqlobject.dberrors.IntegrityError:
    p.write('%n[warning] attachment duplicated: id:' + ¥
            str(attachment_id_counter) + ', key:' + ¥
            ticket_dict[str(r['attach'])])
def copy_attach(p, attach):
    p.write('%n[info] check directory ' + ¥
            self.env.attach_zip + os.sep + attach)
    if os.path.isdir(self.env.attach_zip + ¥
                     os.sep + attach):
        try:
            p.write('%n[info] remove directory:' + ¥
                    self.env.attach_dir + os.sep + attach)
            shutil.rmtree(self.env.attach_dir + ¥
                          os.sep + attach)
        except OSError:
            p.write('%n[warning] remove directory ' + ¥
                    self.env.attach_dir + ¥
                    os.sep + attach + ' was failed')
```

- 本体改造必要だし・・・(ID列無くていいのは小学生までだよなー)

## ■ そこでSQLAlchemy

### ■ PofEAAで言うDataMapperパターン

- ActiveRecord

- モデルとDBが1:1
- ドメインロジックがモデルにある

- DataMapper

- モデルとDBがn:n
- マッピングさせる
  - 非正規化が進んでいる場合有利
  - テーブル構造が煩雑な場合有利

## ■ SQLAlchemy

- 2007年10月に0.4リリース
  - 0.3から文法変わりすぎ
- いつのまにかドキュメントがリリースされていた
  - <http://www.sqlalchemy.org/docs/04/>
- リファレンス、チュートリアルはすごく丁寧だと思う
  - 英語ですが

## ■ こんな感じでいける

```
from sqlalchemy import *
from migrate import *

def upgrade():
    user = Table('user',
                 Column('user_id', Integer, primary_key=True),
                 Column('username', Unicode(50), nullable=False),
                 )

    post = Table('post',
                 Column('post_id', Integer, primary_key=True),
                 Column('user_id', Integer, ForeignKey(user.c.user_id),
                 nullable=False),
                 Column('title', Unicode(250), nullable=False),
                 Column('content', Unicode, nullable=False),
                 )

    user.create(migrate_engine)
    post.create(migrate_engine)

def downgrade():
    migrate_engine.execute("DROP TABLE post")
    migrate_engine.execute("DROP TABLE user")
```

## ■ コントローラにCherryPyを採用

- O/Rよりこちらの方が道が険しいかな

```
from turbogears import expose
```

```
class HardCodedAddition:  
    def cantgohere(self):  
        pass  
    @expose()  
    def twoplustwo(self):  
        return "four"  
    @expose()  
    def fortyplus(self, number):  
        return "40 + %s = %s" % ¥  
        (number, 40 + int(number))
```

```
class AlwaysSeven:  
    @expose()  
    def index(self):  
        return "7"
```

```
class Never404:  
    @expose()  
    def default(self, *args):  
        return "The next parts were %s" % str(args)
```

```
class Root:  
    seven = AlwaysSeven()  
    add = HardCodedAddition()  
    foobar = Never404()  
    @expose()  
    def index(self):  
        return "Welcome"  
    @expose()  
    def fun(self):  
        return "...and yet startlingly productive"
```

## ■ ビューにGenshi (<http://genshi.edgewall.org>) を採用

- 11から標準
- ていうか作ってるのedgewallだし

The screenshot shows the Genshi website homepage. The main heading is "GENSHI" with the tagline "Generate output for the web". Below the heading is a navigation menu with links for Home, Trac, Genshi, Babel, Bitten, and Posterity. A secondary navigation bar includes Wiki, Timeline, Roadmap, Browse Source, Build Status, View Tickets, New Ticket, and Search. The main content area features a sidebar with links for Home, Download, Documentation, Mailing Lists, License, and FAQ. The main text describes Genshi as a Python library for parsing, generating, and processing HTML, XML, or other textual content. It highlights its main feature: a template language that is smart about markup. A list of key features is provided, including intelligent automatic escaping, template directives, independence from a specific serialization format, stream-based filtering, and match templates. A "Quick Links" section lists resources like the Genshi tutorial, frequently asked questions, and performance numbers. To the right of the text is a flowchart illustrating the Genshi architecture. The flowchart is organized into four main stages: parsing, building, filtering, and serialization. The parsing stage includes HTML and XML parsers. The building stage involves creating elements and fragments from a stream, and templating with Markup and Text templates. The filtering stage includes HTML form fillers, sanitizers, I18N translators, and transformers. The serialization stage includes HTML, XML, XHTML, and Plain Text serializers.

**Genshi**  
*Python toolkit for generation of output for the web*

Genshi is a Python library that provides an integrated set of components for parsing, generating, and processing HTML, XML or other textual content for output generation on the web.

The main feature is a **template language** that is smart about markup: unlike conventional template languages that only deal with bytes and (if you're lucky) characters, Genshi knows the difference between tags, attributes, and actual text nodes, and uses that knowledge to your advantage. For example:

- Intelligent automatic escaping greatly reduces the risk of opening up your site to **cross-site scripting** attacks (XSS).
- **Template directives** are often less verbose than those in most other template languages, as they can be attached directly to the elements they act upon.
- **Independence from a specific serialization format** lets you instantly switch between generating well-formed HTML 4.01 and XHTML 1.0 (or other formats).
- **Stream-based filtering** allows you to apply various transformations as a template is being processed, without having to parse and serialize the output again.
- **Match templates** let you enforce a common structure on template output, and **more**. This, in combination with **XInclude** support, is used instead of the more rigid inheritance feature commonly found in other template languages.

For those cases where you don't want to generate markup, Genshi also provides a simple **text-based template language**.

**Quick Links**

- [Genshi tutorial](#)
- [Frequently asked questions](#)
- [What people say about Genshi](#)
- [Some performance numbers](#)
- [Projects extending Genshi](#)

```
graph TD
    subgraph parsing
        HTML[HTML Parser]
        XML[XML Parser]
    end
    subgraph building
        Element[Element]
        Fragment[Fragment]
    end
    subgraph templating
        Markup[MarkupTemplate]
        Text[TextTemplate]
    end
    Stream[Stream]
    subgraph filtering
        HTMLFF[HTML Form Filler]
        HTMLS[HTML Sanitizer]
        I18N[I18N Translator]
        Transformer[Transformer]
    end
    subgraph serialization
        HTMLS[HTML Serializer]
        XMLS[XML Serializer]
        XHTML[XHTML Serializer]
        Plain[Plain Text Serializer]
    end
    HTML --> Stream
    XML --> Stream
    Element --> Stream
    Fragment --> Stream
    Markup --> Stream
    Text --> Stream
    Stream --> HTMLFF
    Stream --> HTMLS
    Stream --> I18N
    Stream --> Transformer
    HTMLFF --> HTMLS
    HTMLS --> XMLS
    I18N --> XHTML
    Transformer --> Plain
```

## ■ AjaxはMochikitで

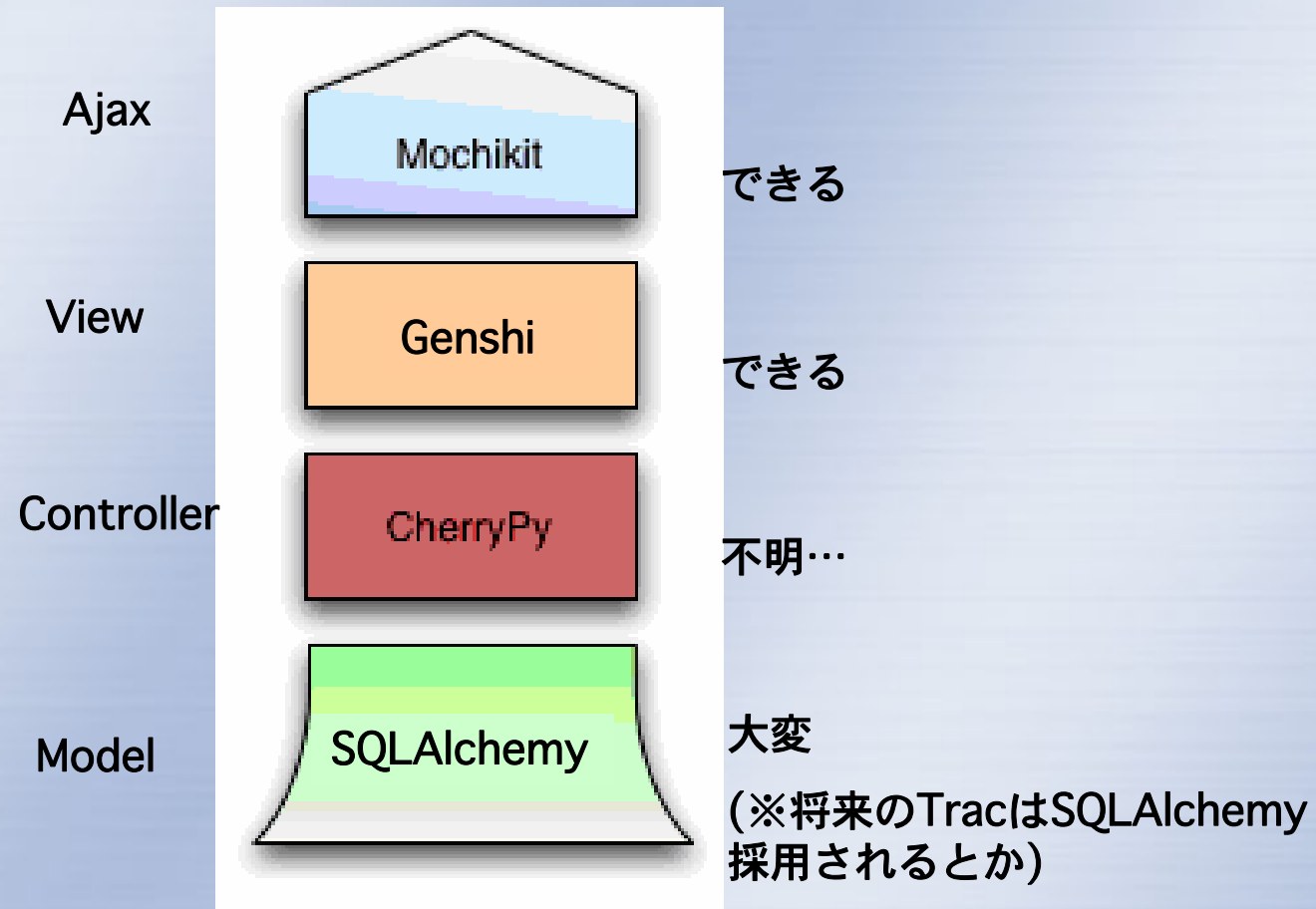
- Twistedにインスパイアされているとか、Pythonと親和性高いとか



The screenshot shows the MochiKit website homepage. At the top left is the MochiKit logo, a small robot character. To its right is a navigation menu with links for HOME, ABOUT, BLOG, and PROJECTS. Below the navigation menu are several links: [Ads by Google](#), [In JavaScript](#), [Date JS](#), [XLS JavaScript](#), and [JavaScript PHP](#). The main content area has a green background with a diagonal line pattern. A large speech bubble contains the text "MochiKit makes JavaScript suck less". Below this are three dark grey buttons with white text and icons: "DOWNLOAD MOCHIKIT" with a download icon, "READ THE DOCS" with a document icon, and "VIEW THE DEMOS" with a mobile device icon. At the bottom, there is a section titled "MochiKit Intro Screencast" with a small video player thumbnail on the left and text on the right. The text describes the screencast as an introduction to MochiKit 1.1 and provides links for [Quicktime](#), [Script](#), and [Archive.org](#). It also mentions that it is confirmed playable with [QuickTime 7](#) and [VLC 0.8.2](#).

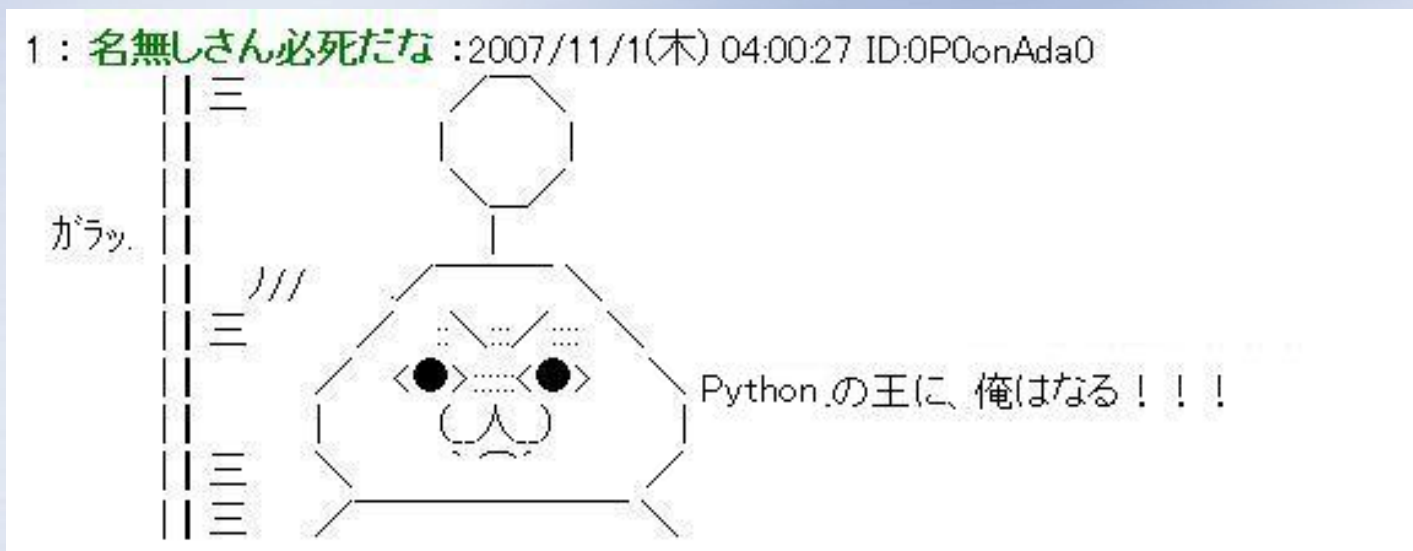
■ TracをTurboGearsライクに

- Webアプリケーションの組み合わせでCMS風に構築できる？
- てかやらないと、あれ以上の本格的な拡張や業務適用は困難？



## ■ 結論

- いけてるフレームワークで開発がしたい！！



- 本家とは別のブランチを作って謹製のソフトウェアを作ろうじゃまいか
- 成果をedgewallにバックポートするぞ！
- 以上、妄想でした

■ ご清聴ありがとうございました