大規模サイトの構築・運用ノウハウ
（仮題）
-GREEの場合-

グリー株式会社
藤本 真樹
<fujimoto@gree.co.jp>

アジェンダなど

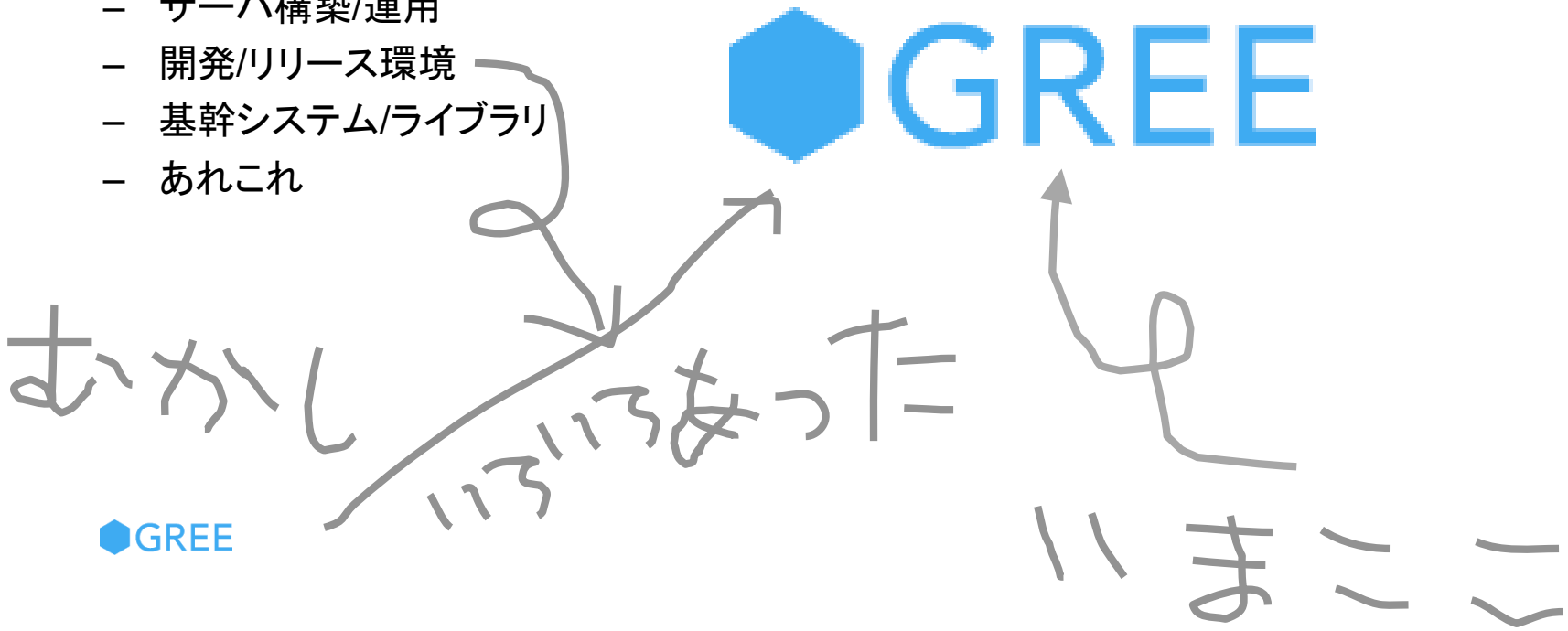
GREE

- 今日のお話がきっとお役にたてる方々
  - インターネット的なサービスを構築/運用されている方々(巨大ではないやつ)
  - これからなんかつくってやろうとたくらんでいる方々
- 今日のお話がきっとお役にたたない方々
  - 金融系システムとかを構築/運用されている方々
  - パッケージソフトウェアを構築されている方々
  - すでに超巨大サービスを構築/運用されている方々
  - お金持ちな方々(いいなー)

アジェンダなど

- GREEの昔(わりと開始当初)の状況
- GREEの現在(まさに今)の状況

- どんなことをしてきたか？
  - ソースコードアーキテクチャ
  - データ
  - サーバ構築/運用
  - 開発/リリース環境
  - 基幹システム/ライブラリ
  - あれこれ



GREE

GREEの昔

GREEの昔

- ページビュー
  - ?万とかそれくらい？(daily)
- ユーザ数
  - ?万〜?万とかそれくらい？
- サーバ台数
  - <10台くらい
- ソースコード
  - 400 files / 65,000 linesくらい
- サービス数
  - 1つ(PC版GREEだけ)

GREEの昔

- 開発チーム
  - 1.5人(2004/12)
- オフィス
  - マンションの一室

開発チームではないふたり

GREE

GREEの現在

GREEの現在

- ページビュー
  - !万〜!万とかそれくらい
- ユーザ数
  - !万人とかそれくらい
- サーバ台数
  - !台〜!台くらい
- ソースコード
  - 5,300 files / 500,000 linesくらい
- サービス数
  - 10くらい

GREEの現在

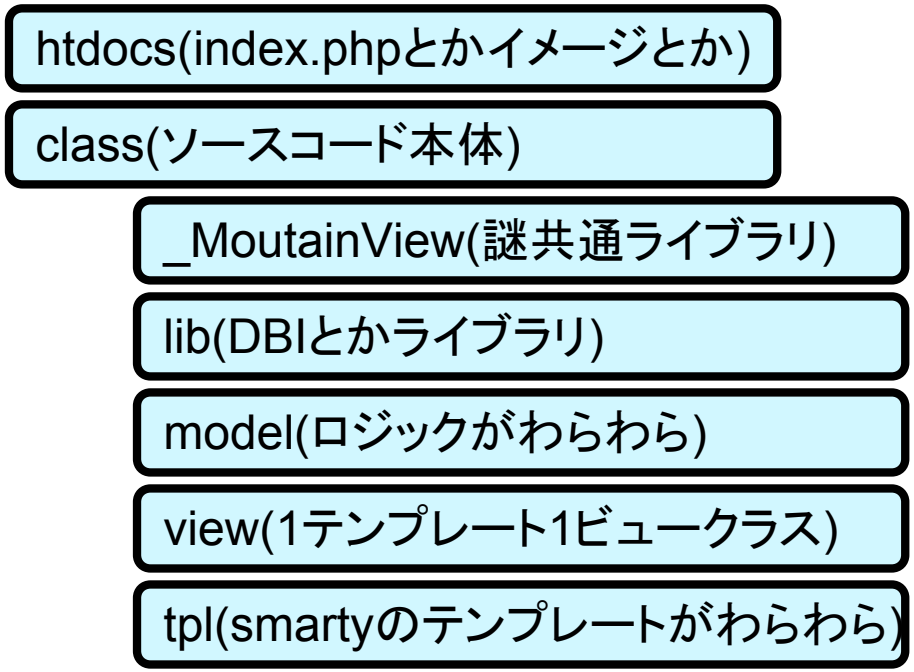
- 開発チーム
  - 20人くらい
- オフィス
  - なんかちゃんとしたところ
  - 2回も引っ越しました
  - ただし、写真はかっこよすぎ

GREE

ソースコード
アーキテクチャ

ソースコードアーキテクチャ

- ディレクトリ構成(その昔)



  - なんか適当なフレームワークっぽいのでうごいてました
  - いろいろ凄いコードがいっぱいありました(switch400行とか)
  - 今にして思えばこのときもっとドラスティックにリファクタリングしておくべきでした

ソースコードアーキテクチャ

GREE

- 僕らになにが必要だったか？
  - フレームワーク移行(現在GREEの全てのサービスはEthnaにのっかっています)
    - 移行に1年半くらいかかりました(あしかけ)
    - もともとMVCっぽい感じになっていたのは多少よかったけど
  - 激しくリファクタリング
    - でもまだ3%くらいもとのコードが残ってます
  - 複数サービスの構築と連携
    - いろんなサービスをつくりたくなります
    - あるいは、既存サービスの1機能を別のサービスとして切り出したくなったりもします
    - 相互に、必要な機能は連携しつつ、依存度を可能な限り下げたいです
    - 一方で、共通な処理のコピペも最小限にしたいです

ソースコードアーキテクチャ

- こんな感じで
  - frontend
  - service
  - src
- の3セクションでGREEの全サービスを構築しています
- その他、例えば値の列挙値を得るために(SELECTボックスで出す値の一覧とか)serviceにはregistryがあったり
- 普通に必要になってくるような処理にはフレームワークを準備していっています

```
$id =& getService('id');
$reg =& $id->getRegistry();
$list = $reg->getKeyList('key');
```

ソースコードアーキテクチャ

- ディレクトリ構成(現在)
  - frontend(各サービス毎)
    - default(各frontendの基底クラス)
    - mpet(例えば、です - ドメイン名とほぼ一緒)
      - ⋮
      - act(ethnaアクションクラス)
      - app(ethnaコントローラや基底クラス)
      - class(frontend固有のクラスライブラリ)
      - cli(コマンドラインなファイル)
      - config(frontend固有設定ファイル)
      - filter(ethnaフィルタクラス)
      - htdocs(document root)
      - lib(smarty pluginとか)

ソースコードアーキテクチャ

GREE

- あとはviewとかtplとかがあります
- っていうのがGREEでは

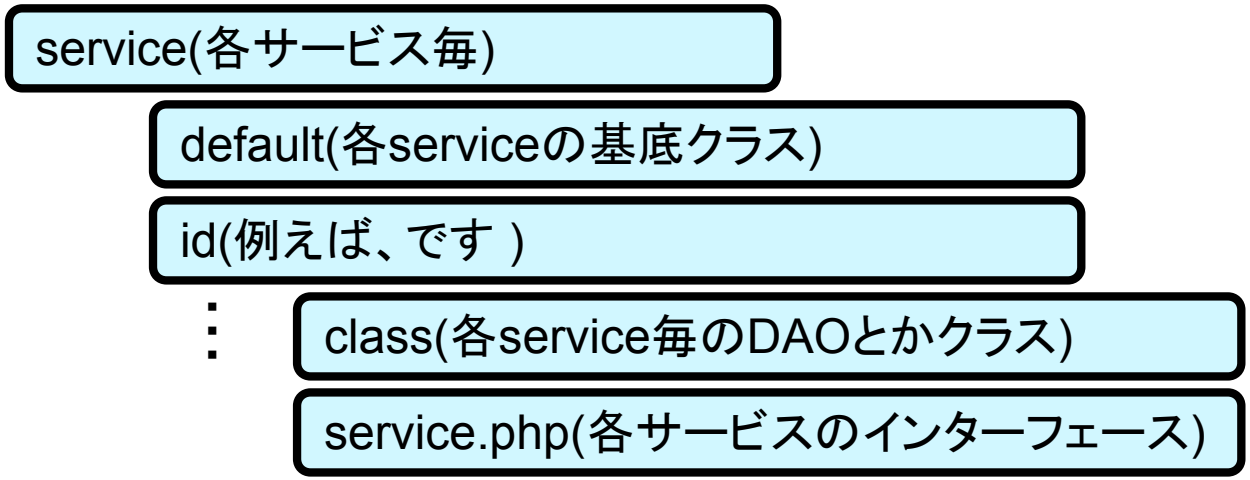
  ethna add-frontend [name]

- とかするとできます

- 基本的にはethna(っていうフレームワーク)にのっとってます
- アクセスログを出力するためのフィルタなどは共通化されていて、基本的にはサービスの構築に集中できる…はず
- あとはsmarty pluginの整理と共通化が課題です(いろいろしがらみが)

ソースコードアーキテクチャ

- ディレクトリ構成(現在)



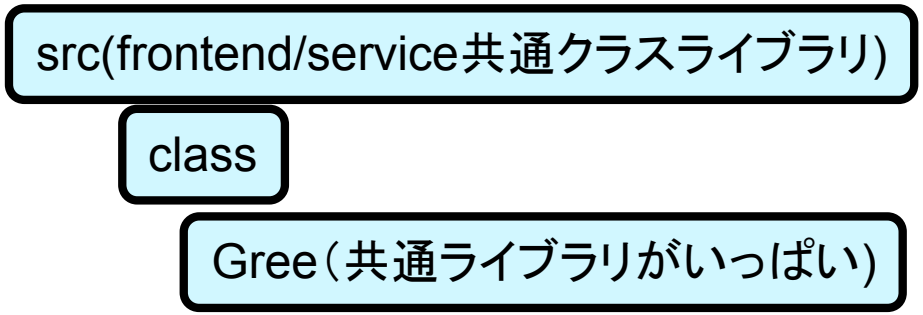
- 各serviceはfrontendに非依存です
- 各frontend(ウェブサービス)は

  $id =& getService('id');

- とかしてserviceのインスタンスを取得して、処理を実行します
- これで、例えば「あっちのサービスからもSNSのあしあとをつけたい」とかいう処理を、違和感なく書けます

ソースコードアーキテクチャ

- ディレクトリ構成(現在)



- ほんとにいろんなライブラリがあります
- つかってるのもあんまりつかってないのも…

ソースコードアーキテクチャ

- bootstrapファイル
  - ディレクトリが増えてきて、構造が複雑になってくると
  - 「あれ？この定数ってここでつかっていいんだっけ？」
  - 「あれ？このメソッドってここで呼んでいいんだっけ？」
  - とかとか出てきます

- だったらrequice_onceしとけばいいじゃん
  - 50,000 filesもあると、require_onceのコストも馬鹿になりません(もちろんeaccelerator とか入れてますが - ちなみにコンパイルキャッシュを切ると相当遅いです)
  - ということでbootstrapファイルとrequireのルール作り

ソースコードアーキテクチャ

- src/Gree_Bootstrap.php
  - 全サービス共通の定数
    - define('PATH_ROOT', dirname(dirname(__FILE__)));
      - とかいうパス定義とか
  - 絶対使うだろー、的なファイルのrequire
    - require_once PATH_SRC_CLASS . 'Gree/Util.php';
      - とか
- でもって
  - 各frontend以下の値は、各frontend毎に勝手に(但しfrontend間の依存は原則禁止)
  - 各service以下のクラス、定数等はgetService('service')を通じて
  - src以下のクラスライブラリはbootstrapでrequireされているもの以外は適宜require
- ってなルールです(書きながら決めた)

ソースコードアーキテクチャ

- 要するに
- 複数のサービスでの機能連携は前提にしておいたほうがよかったなぁ
  - 別に形はどうあれ
  - 実はethnaはその辺あんまり考えられていない
- requireのorderは最初からある程度考えておいたほうがいいです
  - なんか適当にrequireしまくっとけー、というのは何かがヤバい
    - ってそれGREE
- フレームワークの機能を存分に活用するなりして、ウェブアプリケーションの機能をひたすら共通化
  - ここにコストを惜しまないほうがよいと思います

ソースコードアーキテクチャ

- もうちょっと
- どうせリファクタリングは避けられません
  - テストがきちんと揃っていればいいけど…
  - GREEでもなんだかんだで20%くらいのコストはそっちに割いてきたような…気がします(完全感覚値)
- あと、このご時世なので他言語ブリッジも考えておいたほうがいいかもです
  - PHPだけ、とかキツくなっていきているというのが正直なところです
  - Javaとか、最近すげーっすね(JRubyとか)

GREE

データ

データ

- 増え続けるデータとアクセスとの闘いの日々
  - パフォーマンス
    - どんだけ速くデータをとってこられるか
  - スケーラビリティ
    - データが無限に増え続けてもサーバリソースで問題を解決できるか？
  - 似て非なる両者(やることはわりと一緒)

- 当たり前ですが、データは増え続ける一方です
- 増えるペースも増え続けてます

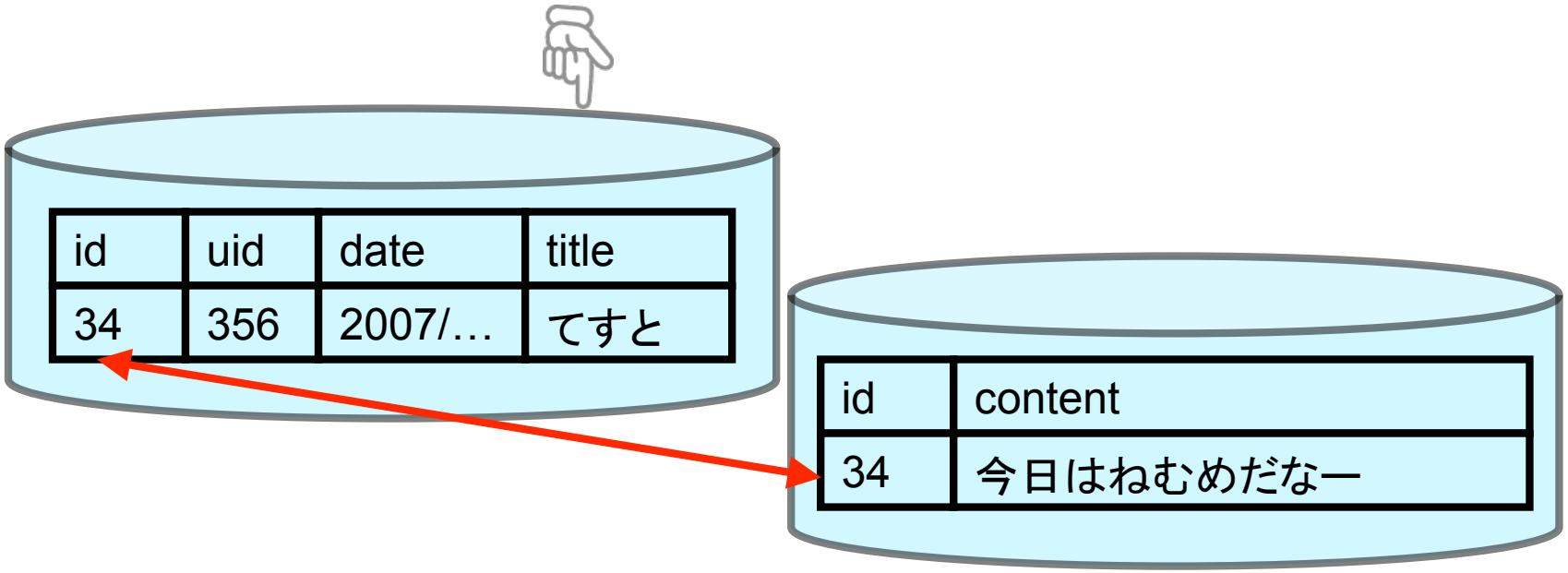
- やることってシンプル
  - データベース分割
  - インデックス張る
  - キャッシュ
    - データキャッシュ
    - スマートキャッシュ(書きながら命名)

データ

- TEXTフィールド分割
  - まず最初にこれがきます(TEXTフィールドがあれば)

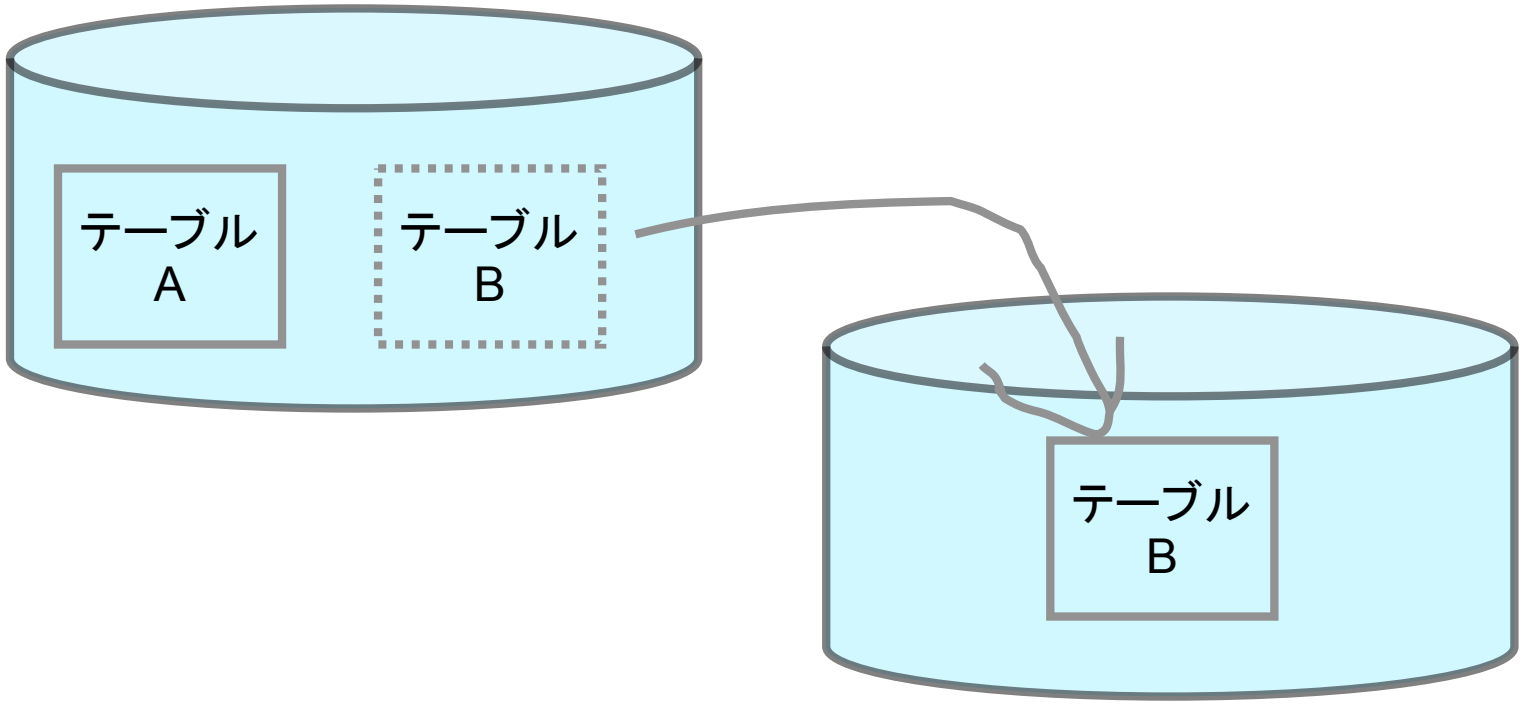
- こんな感じ(blogっぽいテーブル)

id	uid	date	title	content
34	356	2007/…	てすと	今日はねむめだなー



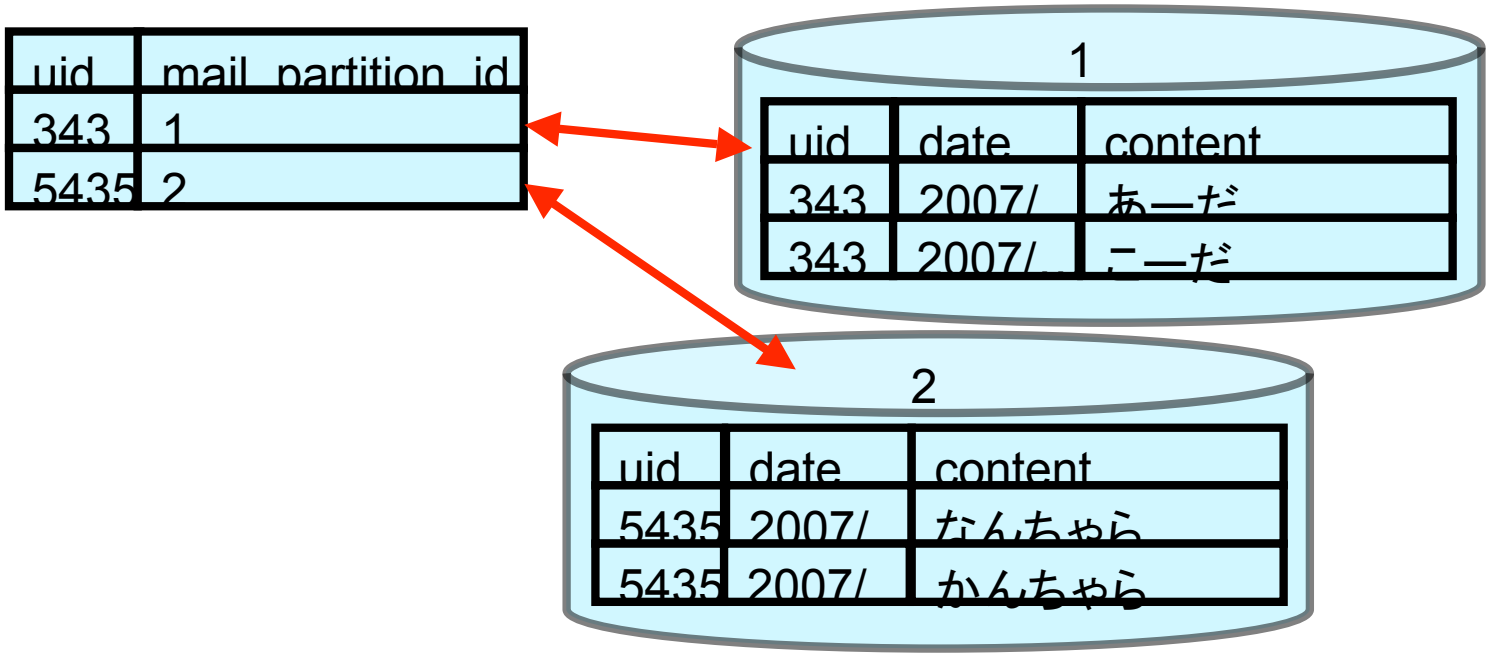
データ

- テーブル分離
  - 同じデータベースにあるJOINしてないテーブル(あるいはしててもJOINを外して)を、別のサーバへ移します
  - ま、移すだけです
- こんな感じ



データ

- テーブル分割
  - 一つのテーブルを複数のテーブルに分割します(パーティショニングと呼んでいるそうです@mixiさん)
  - 詳しくは
    - http://mysqlconf.com/presentations/mysql06/mixi_update.pdf
  - もうやってることはほとんど一緒です
- こんな感じ(GREE内メールっぽいテーブル)



データ

- インデックスを張る
  - slow-logをみてEXPLAINしてみたり
  - データベースインターフェースライブラリでSQLをフックしたり、SQLログを使って全クエリをEXPLAINしたり
  - してひたすらただしいINDEXを模索する日々(あと、そもそもそのSQLなのか、とか？)
- MySQLのINDEXは、うーん

  *SELECT * FROM [table] WHERE foo=bar ORDER BY buz*
- とか、どうがんばってもパフォーマンスが出にくい

データ

- データキャッシュ
  - 普通のキャッシュです
  - キャッシュストレージは
    - Memcache
    - Mysql
  - インターフェースはそろえておいてストレージの切替は簡単(factoryっぽいかんじ)
  - ethnaになんとなくフィードバックしてみました

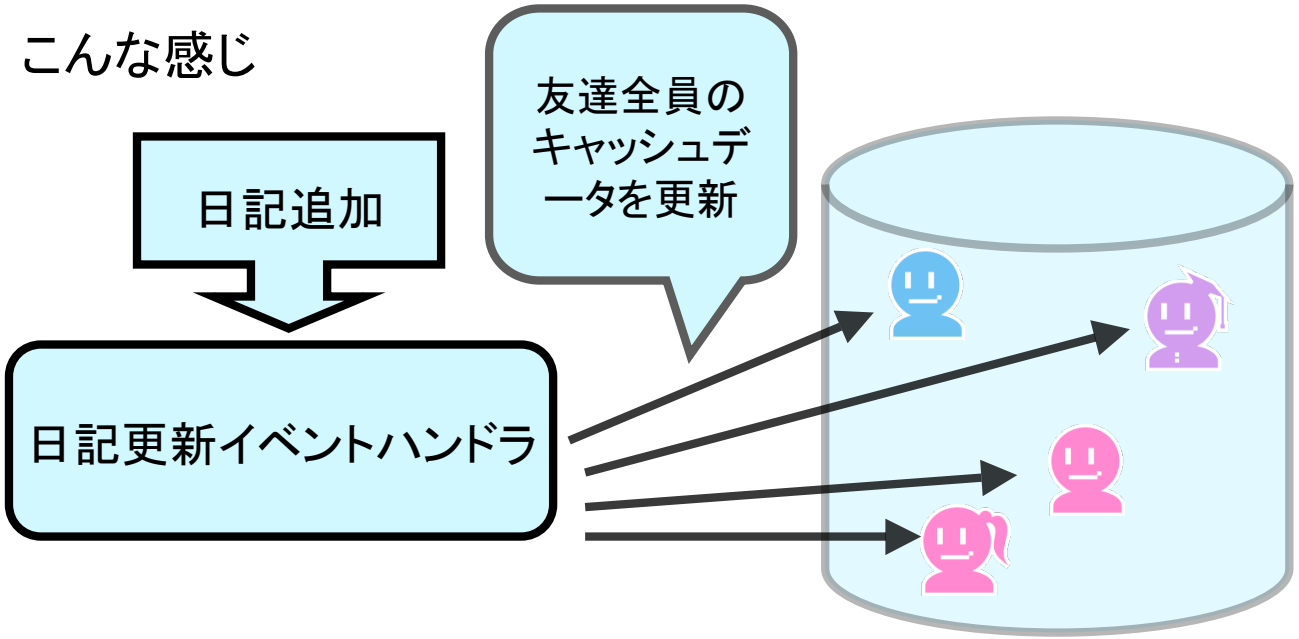
```
$cm =& CacheManager::getInstance('mysql');
$cm->setNamespace('foo');
$cache = $cm->get('bar');
```

- ってなかんじです
  - あんまり更新しなくてよい or されないデータにつかっています
    - 例: 関連キーワード

データ

- スマートキャッシュ
  - SNSとかだと、普通のキャッシュが使えるところが意外にすくないです
  - たとえば「友達の新着日記」
    - 普通に一定時間キャッシュすると新着がすぐ出なくてつまらない
    - 友達が日記書いたらキャッシュをexpire、とかめんどい
  - どうせめんどいなら、日記を書いたら友達の新着キャッシュを更新しちゃえ(キャッシュがない場合だけDBにいけばいいや)

- こんな感じ



データ

- まとめてみると

- TEXTフィールド分離
- テーブル分離
- テーブル分割

- JOINしない
- 正規化がんばりすぎない

- キャッシュも使いどころを考えてうまいこと…
  - Memcacheって、なんか…だめ

- そもそもアプリケーションをなんとか…(SNSっていろいろ面倒)
  - 「ともだちの～」とかACLとか
- 最近はネットワークトラフィックが馬鹿にならないです

GREE

サーバ構築/運用

サーバ構築/運用

- 500台くらい超えてくるとさすがに頭にはいってこないです
  - どんなサーバが何台あるかとか
  - そもそも何台空いてるのかとか
  - 壊れてるとか壊れてないとか
  - どこのラックにどのサーバがはいってるのかとか
  - …
- サーバ管理システムつくりました
  - 意外になかった
  - 監視ツールやステータスグラフツールはいっぱいあるんですけど…

サーバ構築/運用

- こんなふうです

index

server

サーバ検索
サーバー覧
サーバ追加

apt

package一覧
package依存関係グラフ

GREE Server

利用状況(タイプ別)

タイプ	ユニット数	使用中	未使用	取置	月額
lvs					
proxy					
www					
batch					
db					
db(search)					
image					
cache					
storage					
ns					
internal					
staging					
dev					
router/switch					
rack					
合計					

利用状況(ハードウェア別)

ハードウェア	ユニット数	使用中	未使用	取置	月額
db (generation-1/AMD Opetron)					
db (generation-1/Pentium M)					

サーバ構築/運用

GREE

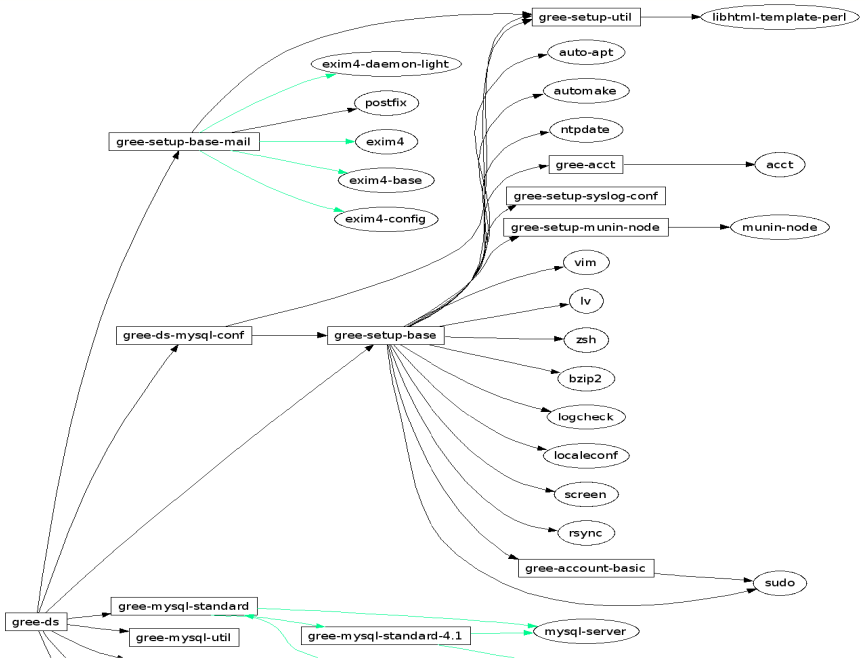
- どんなタイプのサーバが何台空いてるか
- サーバがいつ投入されて、いつ壊れて、いつ直ったか
- などなどサーバ管理でやっぱりあったほうがうれしいです

- アプリケーションでも
  - リリース先サーバの一覧をどこにかこうか？とか
  - 画像のrsync先のリストを直で書きたくないよーとか
  - こんな問題が解決できてちょっとうれしい

- あとは他のツールとのインテグレーションしたい！

サーバ構築/運用

- ついでにpackage依存グラフ
- ichii386作成



サーバ構築/運用

- Debina GNU/Linux完全依存(まだほとんどsarge)
  - 全部aptパッケージ管理

```
$ sudo debian_initialize_server.sh [hostname]
$ sudo aptitude update
$ sudo aptitude install [gree-server-meta-package]
```

  - でサーバ構築完了
  - って、ちょっと美化しすぎ(ほんとはもうちょっとこまごました作業が:)

- いろんなアップデートも

```
$ sudo aptitude update
$ sudo aptitude upgrade
```

  - してまわるだけでおk
  - ってちょっと美化しすぎ

サーバ構築/運用

- つまりですね
- サーバ台数増えてくると、管理ツールっぽいものが必須になってきます(300台こえたくらいで考え始めました)
  - 意外にない(あったらおしえて！)
- 多少の犠牲は払ってでもサーバOSとかは一律でそろえたほうが楽です
  - でもsargeをどうやってetchにするかが目下悩みです
  - でもこれで大分楽してるなーとは思います

GREE

開発/リリース環境

開発/リリース環境

- 開発環境(超初期)
  - ソースコード: 社長のvaioのHDD
  - 秀丸で編集
  - productionサーバへscp
  - 各ウェブサーバでちょっとずつソースコードが違う
    - 衝撃

開発/リリース環境

- 開発環境(初期)
  - とりあえず開発環境用にサーバを1台確保
  - 最新のソースコードをかき集め
  - CVSリポジトリにimport(よかったよかった)
  - ウェブサーバへのリリース用スクリプト作成
    - 単純に1台ずつrsync
  - 開発サーバに、GREEのstaging環境を構築(小さな一歩)
    - とりあえずここで開発

開発/リリース環境

GREE

- 開発環境(中期)
  - 2人以上での開発がはじまりました
  - 開発サーバに/home/devN(N=1-10)という感じでディレクトリ作成
  - 各ディレクトリ毎にソースコードをチェックアウト + vhostsを作成(DBは共通)
  - 開発環境ごとにbranchを切るかどうかで大もめ:)
    - なつかしい
  - 結局まずはtrunk一本で開発
  - 各開発環境でcheck in → staging環境でupdate → チェック → リリース

開発/リリース環境

- 開発環境(現在)
  - 10人以上で開発していく状況になってきました(この間にCVS → SVN)
    - cvs2svnで楽勝(おすすめ)
  - とりあえず開発用サーバが遅い
  - ひたすら遅い
  - svn upしたら10分待ち、とか…
  - 開発環境の設定やパッケージングがぐだぐだでもう1環境作るのも大変
  - というか開発用サーバに2台も3台も使いたくない
  - 開発環境をdebian package化(もう意地で)
    - 個人マシンのvmにdebianをいれて(vmwareとかparallelsとか)、そこに開発環境パッケージをインストール
    - 軽くなってうれしい！
    - 開発用サーバでの開発と99.9% compatible
    - Eclipseとかも結構快適

開発/リリース環境

- 開発環境(現在)
  - そんなこんなしている間にウェブサーバ激増
  - リリーススクリプトも書き直し(結局独自)
    - なんかいいのねっすか
  - N並列でのリリース
  - proxyサーバ連携(リバースプロキシから抜いて、アクセスがなくなってからリリース)
  - あれこれ事故防止のためにちょっとずつ改善中…

開発/リリース環境

- ということで

- 開発環境はがんばってパッケージングしておくといいかも(いろんな設定とかも含めて)
- 各環境毎に異なる設定も、localな設定ファイルから自動生成するような仕組みにしておいたりして

- リリースは…なんかいいのあるんですかねー@php
- 結局自前で開発でした
- Symfonyのとか、どれくらい汎用性あるんでしょ？（試してナシ)
- Capistrano

GREE

基幹システム/ライブラリ

基幹システム/ライブラリ

GREE

- 規模が大きくなればなるほどこの辺の充実具合がスピード感を決めてくる気がする今日この頃です
- GoogleさんとかY!さんとか、やっぱりすごい
- GREEはまだまだこれからです

- いろいろ作ってはいるので、いくつかご紹介です

基幹システム/ライブラリ

GREE

- ウェブサーバクラスタ
  - LVS(w/ keepalived) + mod_proxy_balancerでほぼfix
- RDBMS
  - 冗長化するにはmaster-masterなんですが(MySQL Clusterとかはちょっとまだキツい - 安定性とかハードウェアスペックとか)、ちょっと運用がめんどいです
  - 読み込みの分散はslaveを増やせばいいのはいいとして
  - 書込みの分散は「データ」のとこにあったように、手動で分散させていくしかない今日この頃です
  - 夢のRDBMSとかほしい

基幹システム/ライブラリ

GREE

- Gree_Async
  - 似非drubyみたいなの
- Gree_Datetime
  - 日付統合クラス
- Gree_GenericDAO
  - なんかいい感じのORMっぽいデータアクセスクラス
- Gree_HTTP_Loader
  - PEARのHTTP_Requestがばぐばぐだったので…
  - WebDAV通信で主に使ってます
- Gree_Media_Flash
  - Flashであれこれ

基幹システム/ライブラリ

- Gree_RichText_Renderer
  - HTML(+α)タグレンダラクラス(w/ php拡張モジュール)
- モバイルUIフレームワーク
- いくつかのapacheモジュール(QRコード生成、イメージアクセストークンなど)
- その他あれこれクラスライブラリがありますが、この辺はかなり頻繁に使われてます
  - っていうことはきっとみなさんも必要だと思うので、一応…
- あとはEthnaにSmartyとPear
- Smarty遅い

GREE

まだまだ

あれこれ

まだまだあれこれ

- ログ/データマイニング
  - (このあと登場)ichii386に聞いてください
- 広告配信
- 全文検索
- 課金
- コンテンツ監視
- バックエンドツールいろいろ

- 技術者募集(はげしく)

GREE

Thank You For Listening