

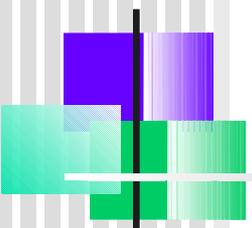


# XML Consortium

## 気象庁防災情報XMLを Google App Engineで 大量配信する一手法

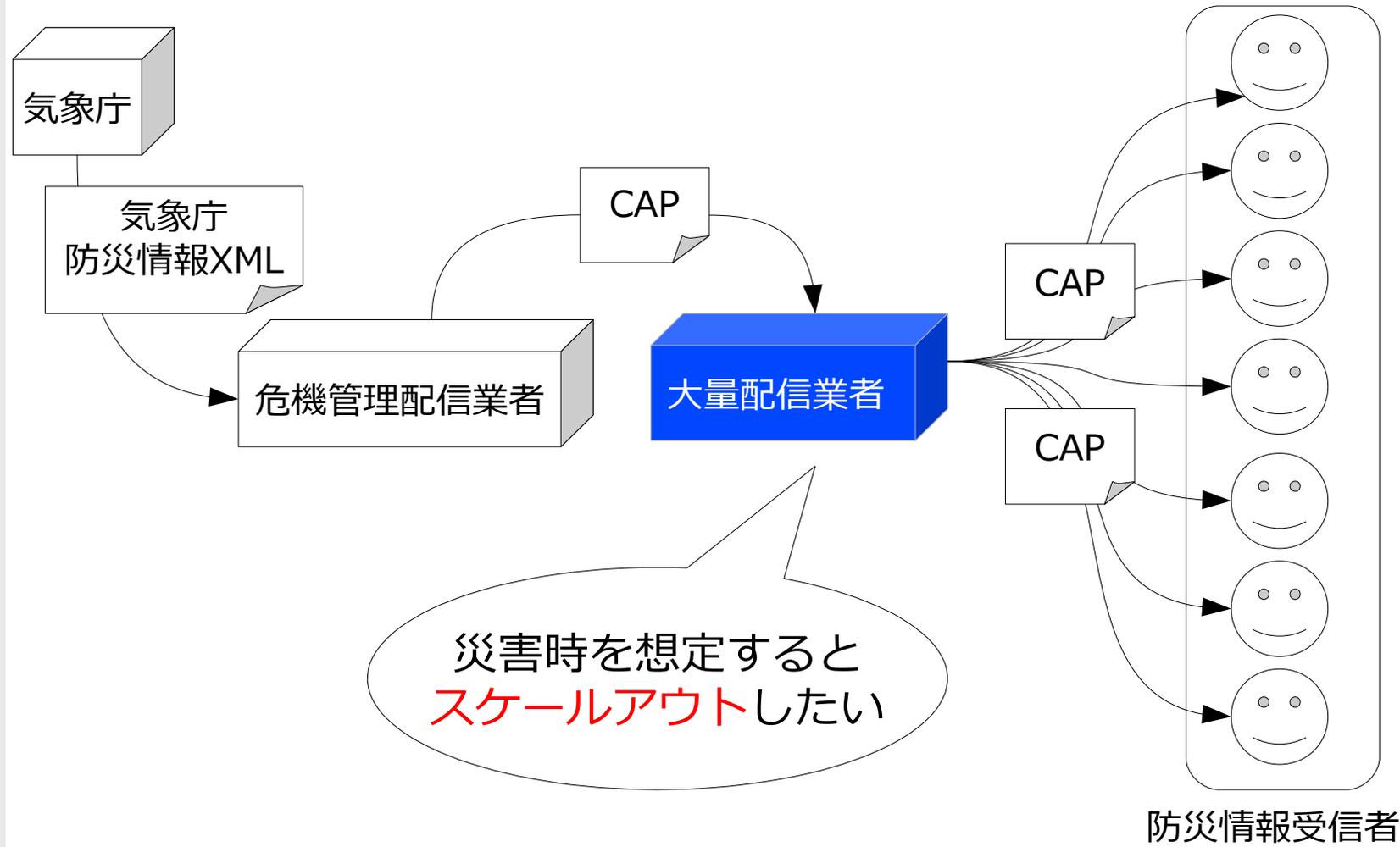
2010年3月18日

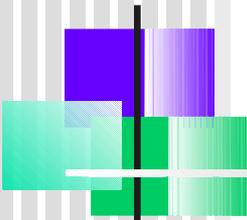
高橋 公一



# 大量配信とは

- より多くの人に防災情報を届ける





# Google App Engine



The screenshot shows the Japanese version of the Google App Engine website. Two callouts are present:

- 自動スケーリング** (Automatic Scaling): A callout bubble pointing to the text "App Engine が多くの企業に選ばれている理由は以下のとおりです。" and the list of benefits.
- インフラの信頼性** (Infrastructure Reliability): A callout bubble pointing to the link "Google インフラの信頼性、パフォーマンス、セキュリティ" in the list of benefits.

**App Engine を使用するメリット**

Google App Engine を使用すると、通常の Web アプリケーションに使われているスケーラブルなシステムをベースとして、独自の Web アプリケーションを構築できます。App Engine アプリケーションは、構築と維持管理も簡単です。またトラフィックやデータストレージの増大に合わせて容易なスケーリングが可能です。App Engine では、サーバーを維持管理する必要もありません。アプリケーションをアップロードするだけで、すぐユーザーが利用できるようになります。

App Engine が多くの企業に選ばれている理由は以下のとおりです。

- [導入が簡単](#)
- [自動スケーリング](#)
- [Google インフラの信頼性、パフォーマンス、セキュリティ](#)
- [コスト効率の高いホスティング](#)
- [リスクのない試用期間](#)

**導入が簡単**

App Engine は、幅広く利用されている技術をベースとして Web アプリケーションを構築してホストするための総合開発スタックです。App Engine では、アプリケーションコードを記述してローカル マシンでのテストが完了したら、クリック操作やコマンドラインスクリプトで簡単に Google にアップロードできます。Google にアップロードしたアプリケーションは Google によってホストされ、そのスケーリングも自動的に行われます。つまり、システム管理、アプリケーションで使用するインスタンスの生成、データベースのシャーディング、新しいマシンの購入などについて心配する必要はないということです。維持管理はすべて Google に任せられることができるため、開発者はユーザーが求める機能の拡充に集中できます。

- マルチキャスト(ブロードキャスト)  
最も適しているが、Google App Engine ではUDPが使用できない
- メール  
プッシュ型でないため不向き
- HTTP  
必要条件は満たしている

# Task Queue を使う



The Task Queue Java API - Google App Engine - Google Code - Mozilla Firefox

http://code.google.com/intl/ja/appengine/docs/java/taskqueue/

Google code

4,000 developers, 150 companies, 80 sessions. [Register for Google I/O!](#)

Google App Engine

Home Docs FAQ Articles Blog Community Terms Download

Downloads

[System Status](#)

[Issue Tracker](#)

Getting Started

[What Is Google App Engine?](#)

Java

Python

Java

[Overview](#)

[Servlet Environment](#)

[Storing Data](#)

Services

[Memcache](#)

[URL Fetch](#)

[Mail](#)

[XMPP](#)

[Images](#)

[Google Accounts](#)

[Task Queues](#)  
(Experimental)

## The Task Queue Java API

App Engine applications can perform background processing by inserting tasks (modeled as web hooks) into a queue. App Engine will detect the presence of new, ready-to-execute tasks and automatically dispatch them for execution, subject to scheduling criteria.

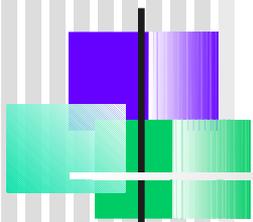
The Task Queue service is currently released as an experimental feature. This means the API and behavior of the service may change in ways that are not compatible with earlier releases, even for minor releases of the runtime environments. Please try out this feature and let us know what you think! ([more info](#))

This reference describes the Java API for the Task Queue service. It has the following sections:

- [Overview](#)
- [API Reference](#)

完了

スレッドに代わるもの

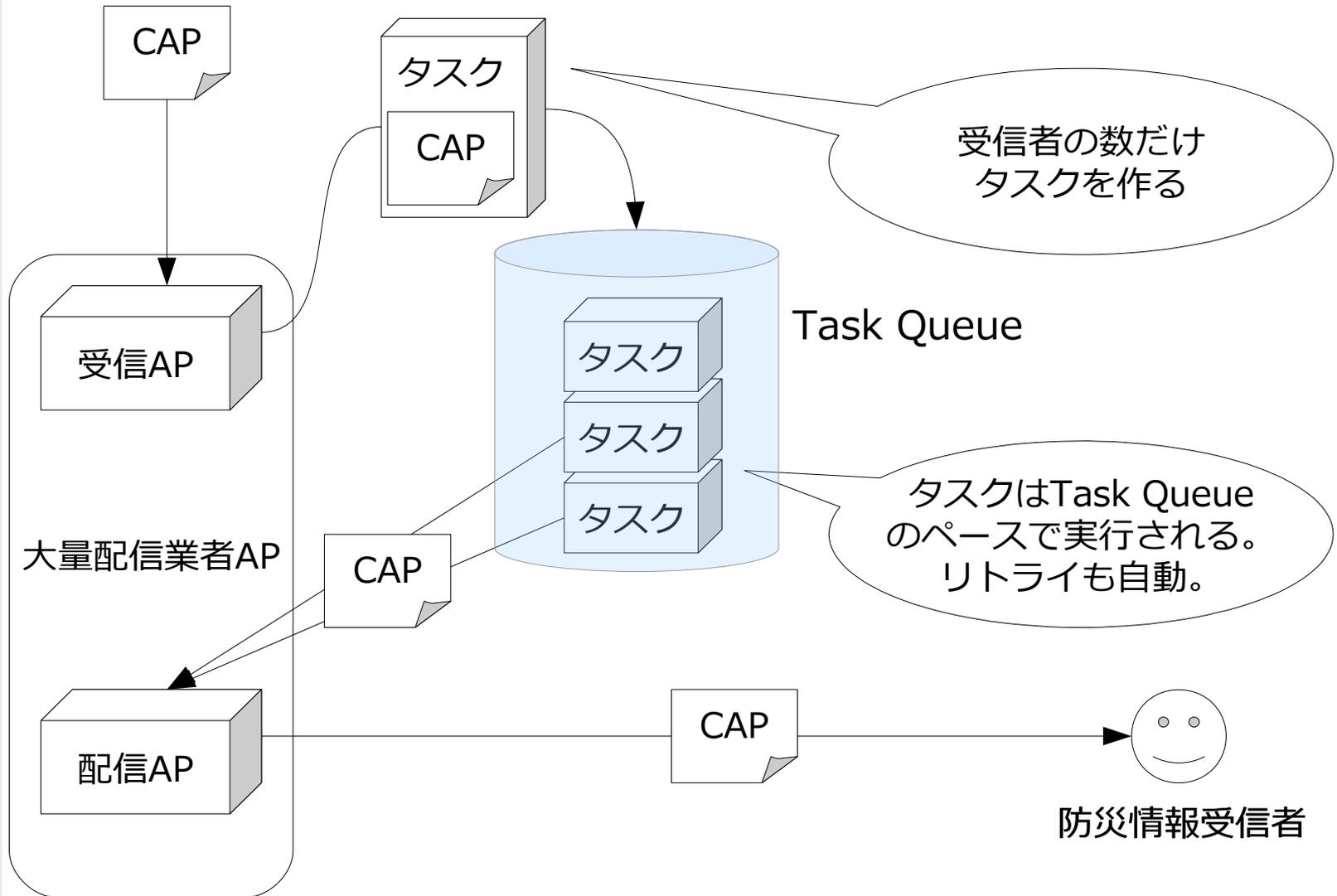


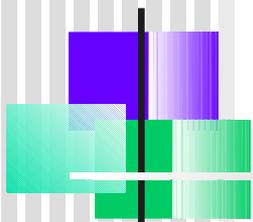
# Task Queue とは



- GAEアプリケーションにHTTPでアクセスするタスクをキューイングする
- タスクを作成した呼び出し側とは非同期にタスクを実行する
- タスクを実行してから30秒程度以内にステータスコードが200番台のレスポンスが返ってこなければ、自動的にリトライし続ける

# Task Queue のイメージ





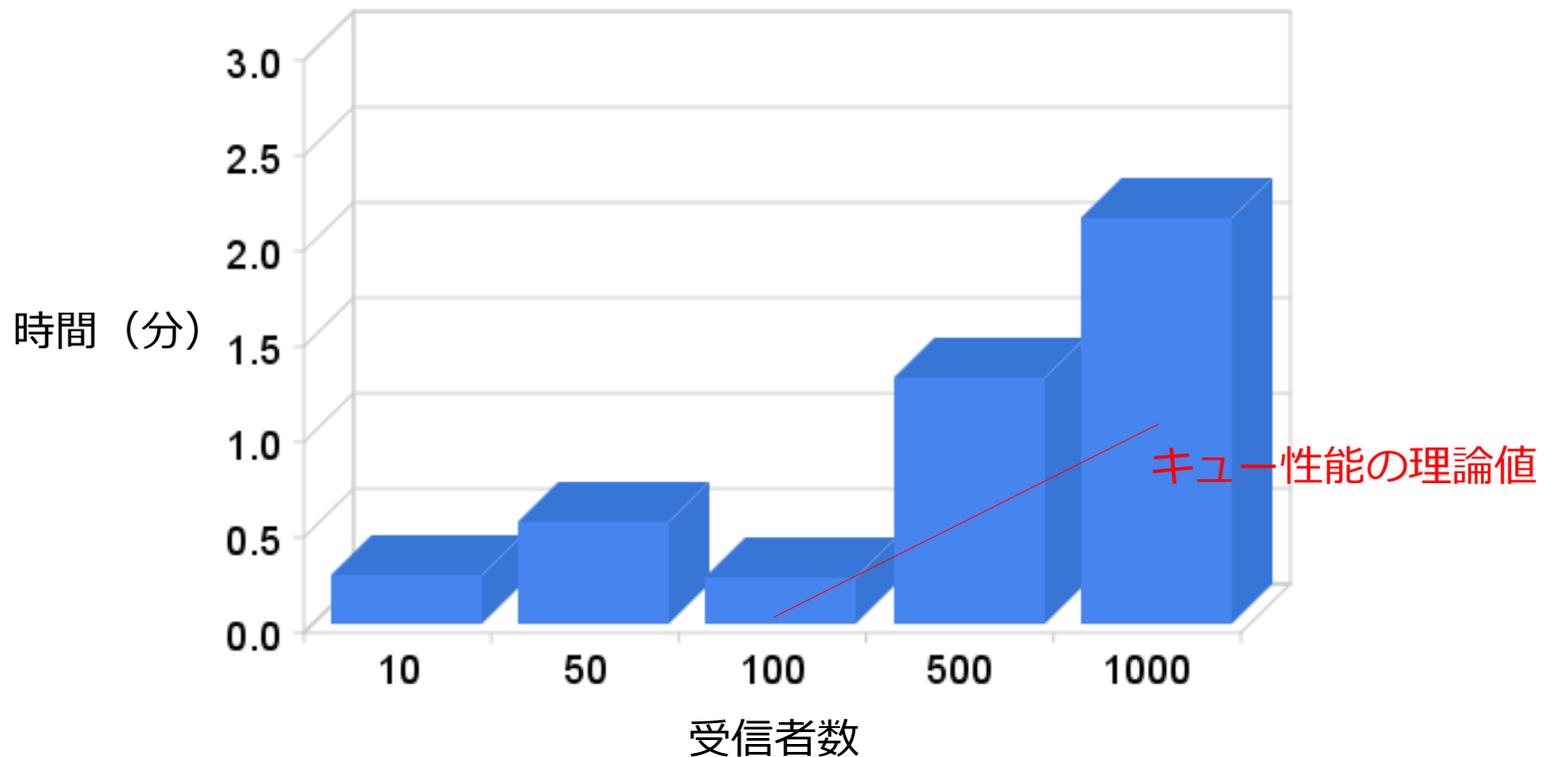
# Task Queue の制限

1 タスクのオブジェクトサイズ	最大10KB
キューの数	最大10個
同時に実行できるタスクの数	最大 毎秒20個 (すべてのキューを合わせて)
タスクの残存時間	最大30日

- その他に、リクエスト数、CPU時間、送受信に使用する帯域に制限がある

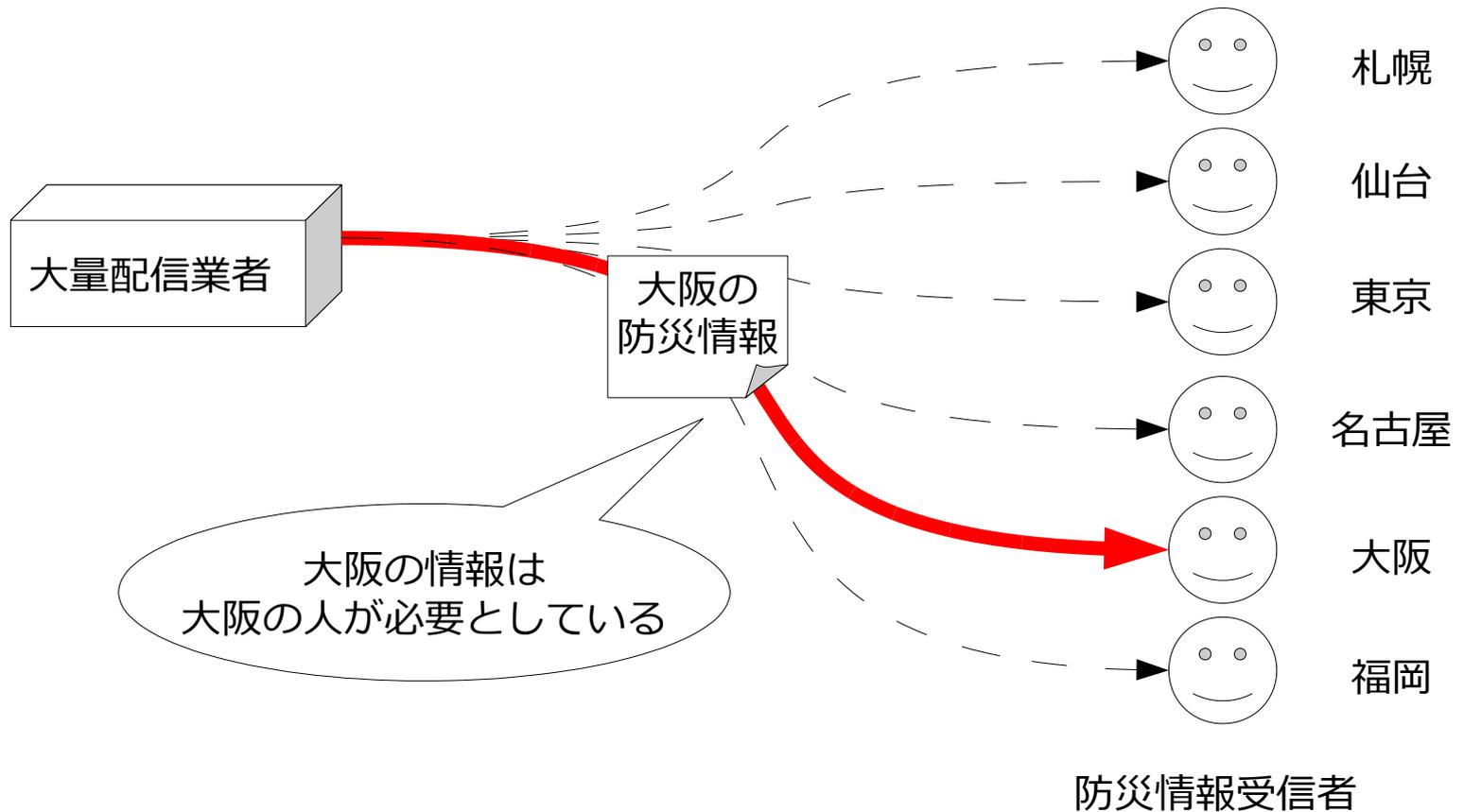
# 性能

- 受信アプリケーションがCAPを受け取ってから、全ての受信者が防災情報を受信し終えるまでの時間を計測（5回の平均）



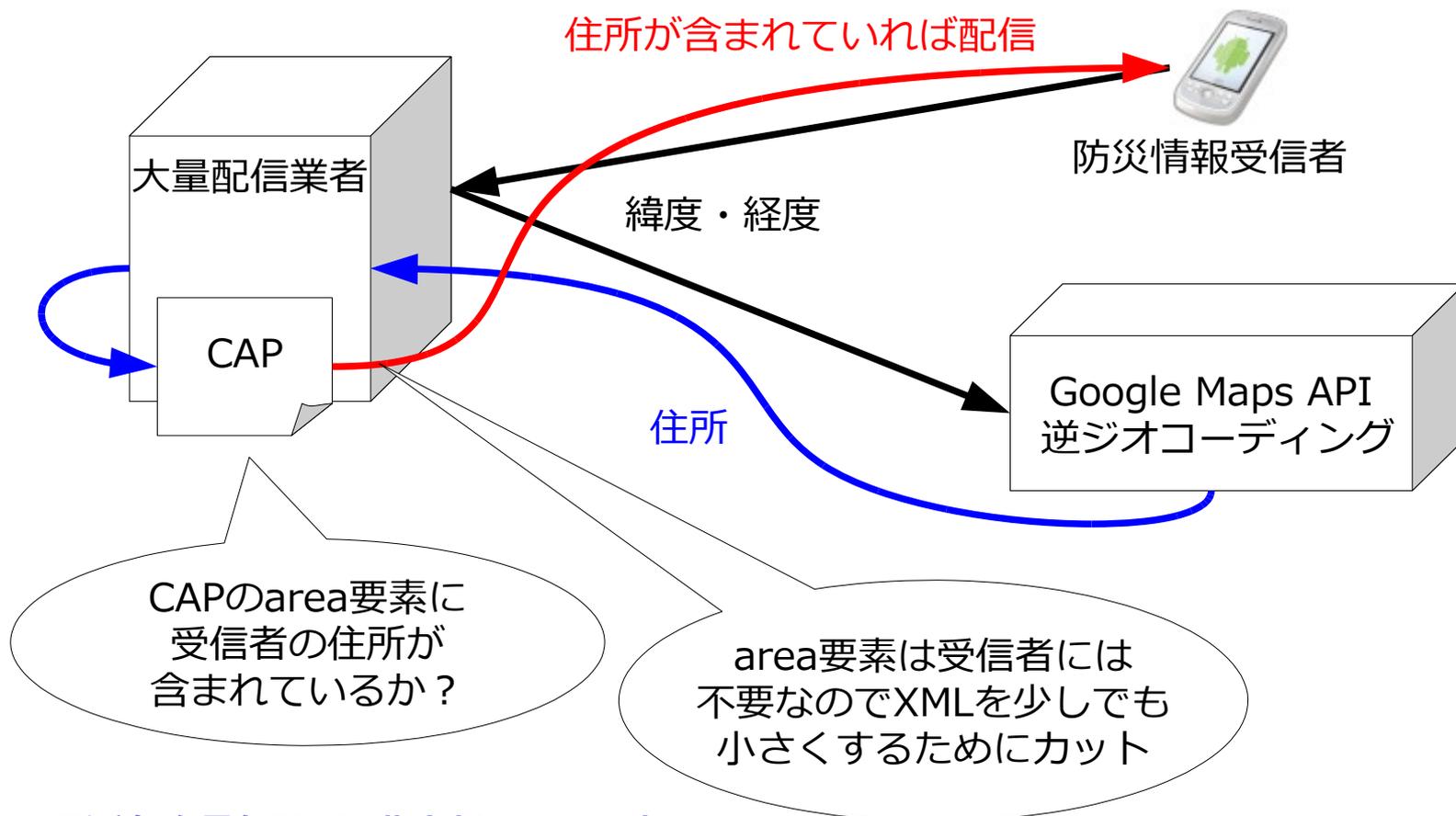
# 位置によるフィルタリング

- 受信者のニーズに合った防災情報を届ける



# 逆ジオコーディングを使う

- 防災情報の対象地域が受信者の現在地と一致するかを判断



- 緯度、経度を住所に変換すること
- Google Maps API にも含まれている
  - クエリーに緯度・経度を含めてHTTP - GET でアクセスすれば、XMLで住所が取得できる

```
http://maps.google.com/maps/geo?  
q=緯度経度  
&output=xml  
&sensor=true  
&hl=ja  
&oe=UTF8  
&key=API キー
```

- あいまいなエリア表現は逆ジオコーディングで取得した住所とのマッチングが難しい
  - 中国地方…気象庁では山口県を九州北部地方に入れている
  - 伊豆諸島南部…何島が含まれるか知っていますか？
  - 静岡県伊豆
- 従来から使われているあいまいなエリア表現は、それに該当する住所に変換してから防災情報XMLに入れるようにすれば、受信者は扱いやすくなる

# 実装のポイント①

- 呼び出し側（危機管理配信業者）の拘束を最小限にする
  - Google App Engine for Java は Servlet 2.5 なので Servlet 3.0 の非同期機能はまだ使えない

```
protected void doPost(HttpServletRequest req, HttpServletResponse resp)  
throws ServletException, IOException {
```

```
    resp.setStatus(HttpServletResponse.SC_ACCEPTED);  
    resp.flushBuffer();
```

～リクエストメッセージの処理～



## 実装のポイント②

- Task Queue のタスクがアクセスする URL には自身のアプリケーションルートからのパスを指定することに注意

```
protected void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
```

```
~
```

```
byte[] payload = ~;
```

```
Map headers = ~;
```

```
String url = "/delivery/for_queue_task";
```



```
Queue queue = QueueFactory.getDefaultQueue();
```

```
TaskOptions options
```

```
    = TaskOptions.Builder.method(TaskOptions.Method.POST);
```

```
options.headers(headers);
```

```
options.payload(payload, req.getContentType());
```

```
options.url(url);
```

```
queue.add(options);
```

```
~
```

- タスクの残留やリトライ多発に注意
  - URI Fetchの低レベルAPIを使うと、タイムアウトを調節できたり、レスポンスを受け取らないように設定できるが、やりすぎるとリトライが多発してパフォーマンスが落ちる
  - 失敗したタスクがキューに残り続けた場合、30日経ってタスクが自動消滅するのを待つしかないため、実質手動で消すものと思っておく必要がある。消えないタスクを放置してリソースを使い果たさないよう、注意！

## 実装のポイント④

- 配備記述子の性能に関わる設定を忘れずにする
  - appengine-web.xml

```
<precompilation-enabled>true</precompilation-enabled>
```

- queue.xml

```
<queue>  
  <name>default</name>  
  <rate>20/s</rate>  
</queue>
```



- 紹介した大量配信業者の実装（ソースコード）は次のURLからダウンロードできます
  - <http://asm-koichi.appspot.com/>

END