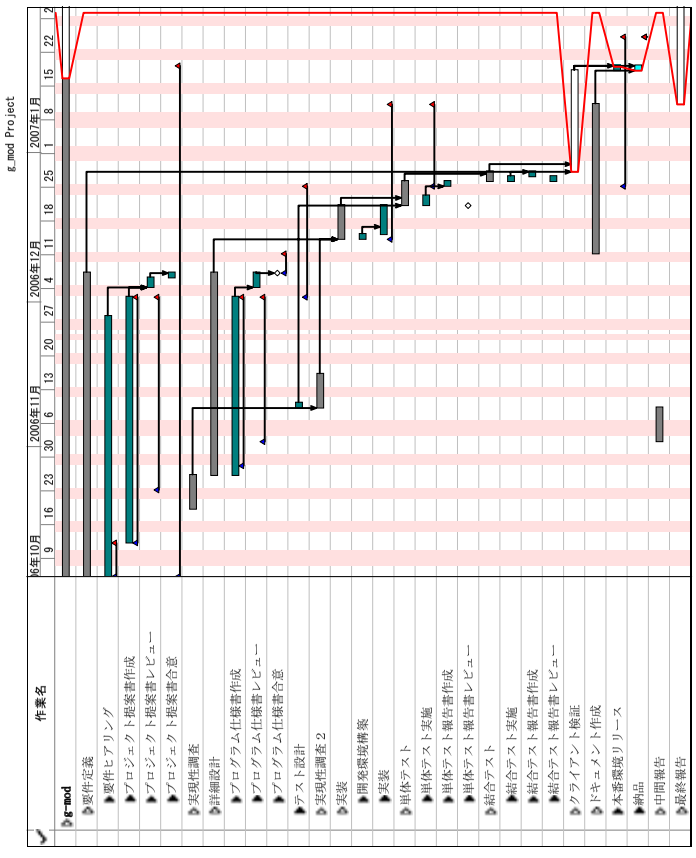
G-mod チーム付録

付録･目次



Geocoding Module　プロジェクト提案書

2006 年 1 月 16 日 1.3 版作成
慶應義塾大学 大岩研究会 g-mod チーム

承認印

版	作成日	作成者	承認者	更新箇所
0.1	2006/10/24	佐藤俊之	－	－
0.2	2006/10/28	佐藤俊之	－	－
0.3	2006/11/06	佐藤俊之	－	－
1.0	2006/11/09	谷田部学	－	－
1.1	2006/12/06	佐藤俊之	－	・ プロジェクトの目的及び成功条件の記載 ・ プロジェクトの目標と品質条件の記載 ・ プロジェクト体制の追加 ・ スケジュールと開発環境の更新
1.2	2006/12/20	佐藤俊之	－	細部修正
1.3	2006/1/16	佐藤俊之	－	細部修正

目次

Ⅰ.　プロジェクトの目的

g-mod チームの目的は，慶應義塾大学デジタルメディアコンテンツ機構様（以降，DMC 様）のデジタルメディアコンテンツ統合システム（以降，DMC システムと記す）で使ってもらえる新しいシステムを開発することである．

上記の「新しいシステム」として DMC システムの新機能である Map 型 UI の一部である Geocoding Module を開発する．
Map 型 UI は，コンテンツ郡をそれらが持つ地理情報を利用して地図上に再配置させる機能を持っている．
その地図に表示するための地理情報を取得することが GeocodingModule の目的である．
GeocodingModule 作成以前は手動でデータを入力していたが，その手間を省くのと正確性を上げるため，本モジュールが必要となった．
用語の定義については 2006 年 11 月 1 日付「DMC ネットワークシステム Map 型 UI 表示機能 機能仕様書」（版 2.1）参照．

プロジェクトの成功条件は，GeocodingModule が DMC システムで使われる際，何らかの報酬を頂くこととする．
報酬の案は現在検討中であり，プロジェクト終了までに DMC 様に提案し，決定することとする．

II. プロジェクトの目標

g-mod チームが先述の目標を達成する上での目標は「品質の高いシステムを構築する」ということである．
目標達成の基準として，本書類に記してある品質基準の達成の他に，以下の項目のうち半数を達成することを挙げる．
以下は暫定的なものであり，プロジェクト完了までに最終的な品質基準項目一覧を作成する．

1. Geocoding Module からのレスポンスは 5 秒以内である
2. クライアント指定期間（1 ヶ月間）誤動作を起こさない
3. インストール，設定ガイドが作成され，g-mod チームがいなくても設定が可能である
4. インストールが簡単であり，3 分以内にインストールすることが可能である
5. 汎用性を高めるため，設定ファイルの設定項目を 20 個以上設定する
6. システムリソースを 2 万キロバイト以内とする

III. 機能仕様

本モジュールの機能仕様については，本書類作成時において最新版である，2006 年 11 月 1 日付「DMC ネットワークシステム Map 型 UI 表示機能 機能仕様書」（版 2.1）内，「IV.3 Geocoding Module の機能仕様」を参照．

IV. 要求定義

本モジュールに対する要求仕様については，本書類作成時において最新版である，2006 年 10 月 4 日付けの「Geocoding Module 要求仕様説明書」（版 1.0）を参照．

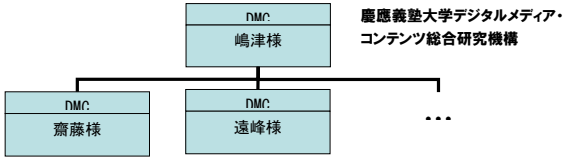
V. ステークホルダー

本プロジェクトの活動におけるステークホルダーは下記の通りである．

クライアント	慶應義塾大学デジタルメディアコンテンツ機構様
システムの開発者	PM：谷田部学　（NextWare 株式会社） SE/PG：佐藤俊之　（慶應義塾大学　環境情報学部四年） SE/PG：平澤秀幸　（同　四年） SE/PG：松田航　（同　三年）
システムの保守・管理・運営	慶應義塾大学デジタルメディアコンテンツ機構様
システムのユーザ	直接的ユーザ･･･慶應義塾大学デジタルメディアコンテンツ機構様 間接的ユーザ･･･DMC システム利用者

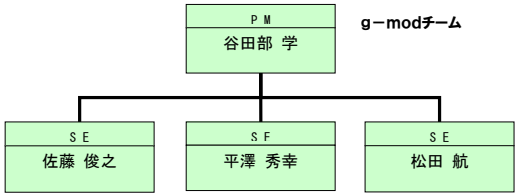
プロジェクトの体制図は以下の通りである．

◆クライアント



※遠峰様が GeocodingModule 開発の窓口となっておられています．

◆システム開発者



VI. 作業範囲

納品物（後述）の**納品及びモジュールの初回導入を行うまで**をプロジェクトの作業範囲として定義する．
本プロジェクトで行う作業の担当割り振りは以下の通りとする．

No.	名称	担当	
		DMC 様	g-mod
1	要求仕様	○	
2	機能仕様	○	
3	プロジェクト定義		○
4	プログラム仕様		○
5	開発環境構築		○
6	結合テスト環境構築		○
7	キャッシュテーブルのレイアウト		○
8	開発		○
9	テスト（単体，結合）		○
10	テスト報告（単体，結合）		○
11	ドキュメント作成		○
12	テスト結果検証		○
13	本番環境配備　(※初回のみ)		○
14	テスト環境配備		○
15	本番運用	○	
16	保守	○	

上記の「5.開発環境構築」について， DMC 様と g-mod 双方合意の上で開発環境を決定する要素がある．
以下のものに対しては，決定したバージョン及び形式に沿ってプロジェクトを進行する．

No.	名称
1	使用する言語のバージョン
2	使用データベース
3	出力及び入力のエンコード

VII. 開発環境

以下の開発環境で実装を行う．

使用言語：JDK 1.5.10
実装環境：Eclipse 3.2.1
座標検索 API：指定された Geocoding API （http://www.geocoding.jp/api/）を用いる
外部ファイル：テキストファイル（設定ファイル，プログラムとリクエストログの形式．拡張子ではない）
データベース：PostgreSQL 8.1.5
出力及び入力のエンコード：UTF-8

VIII.品質基準

VI-1.規約

コーディングの規約は，電通国際情報サービス社による規約（添付：JavaCodingStandard2004.pdf 参照）を用い，
この規約にできる限り沿うこととする．

VI-2.レビュー

下記の工程にてレビューを行なう事とする．

（1） プロジェクト提案書（g-mod 内部レビュー）
（2） 詳細設計書（g-mod 内部レビュー）
（3） テスト方式設計書（g-mod 内部レビュー）
（4） 単体テスト仕様書（DMC 様レビュー，g-mod 内部レビュー）
（5） 結合テスト仕様書（DMC 様レビュー，g-mod 内部レビュー）
（6） 単体テスト結果報告書（DMC 様レビュー，g-mod 内部レビュー）
（7） 結合テスト結果報告書（DMC 様レビュー，g-mod 内部レビュー）

VI-3.テスト

VI-2-1 テスト仕様

単体・結合共に，テスト仕様前に DMC 様による承認を得てからテストを行うこととする

VI-2-2.レビュー

単体・結合問わず，全てのテストに置いて，テスト結果を記録する．
品質基準を達成しているかは，テスト仕様が受理された後でテストを達成することが判断基準となる

VI-2-3.テスト自動化

単体テストにおいては，可能な限りテストは，テスティングフレームワークである JUnit を用いて行うこととする．

IX. スケジュール

本プロジェクトの期間は 2006 年 10 月 1 日から 2007 年 2 月 24 日の間とする．

2006 年 12 月 13 日時点での詳細スケジュールは以下のようになっている．

X. 納品物

本プロジェクトに対する納品物は以下の通りとなっている．

No.	名称
1	プロジェクト提案書
2	プログラム仕様書
3	プログラムソース
4	実行モジュール
5	単体テスト仕様書
6	単体テスト結果報告書
7	結合テスト仕様書
8	結合テスト結果報告書
9	インストール手順書
10	稼動手順書

Geocoding Module　プログラム仕様書

2006 年 1 月 16 日作成
慶應義塾大学 大岩研究会 g-mod チーム

版	作成日	作成者	承認者	更新箇所
0.1	2006/11/02	平澤秀幸	－	－
0.2	2006/11/29	平澤秀幸	－	ドキュメント名称を「プログラム仕様書」に変更 「VI. ユースケース図」を削除 「III. プログラム処理概要」を追加 「IV. データベース構造図」の名称を「IV. 地理情報キャッシュテーブル」に変更 ドキュメント内容の各項目を更新
0.3	2006/12/03	平澤秀幸	－	目次項目の順番を変更 「I. 用語一覧」を追加 「VIII. エラー処理概要」を追加 「IX. プログラムログ」を追加 「X. リクエストログ」を追加 「III. システム構造」の名称を「III. Module 構造」に変更 「II. 概要」の内容を更新 「III. Module 構造」の内容を更新 「IV. 開発環境」の内容を更新 「V. プログラム処理概要」の内容を更新 「VI. 地理情報キャッシュテーブル」の内容を更新 「VII. 設定ファイル」の内容を更新 図 1 と図 2 を更新
0.4	2006/12/06	平澤秀幸	－	「XI. シーケンス図」と「XII. フローチャート図」を削除 「VII. 設定ファイル」の内容を更新

				「IX. プログラムログ」の内容を更新 「X. リクエストログ」の内容を更新
0.5	2006/12/12	平澤秀幸	－	「VI. 入出力」を追加 「VII. アクティビティ図」を追加 「VIII. シーケンス図」を追加 「I. 用語一覧」の内容を更新 「IX. 地理情報キャッシュテーブル」の内容を更新 「XIII. エラー処理概要」の内容を更新
0.6	2006/12/19	平澤秀幸	－	「開発環境」を削除 「IV. 入出力」の内容を更新 「VIII. 地理情報キャッシュテーブル」の内容を更新 「XII. エラー処理概要」の内容を更新
1.0	2007/01/16	佐藤俊之		テーブル設計の内容を更新 HCP チャートの内容を更新
1.1	2007/01/22	佐藤俊之		松澤さんレビューに伴う処理の流れの変更 HCP チャートの内容を変更 設定ファイルの内容を更新 入出力を更新 シーケンス図を更新 アクティビティ図を削除 クラス図を追加 システムログを更新 アクセスログを更新

				エラー処理を更新

概要

Geocoding Module は、DMC ネットワークシステムの新機能である Map 型 UI の一部である。渡された地名情報に該当する緯度経度情報を返す。Geocoding Modulle の位置づけは図 1 と「DMC ネットワークシステム Map 型 UI 表示機能　機能仕様書」を参照。

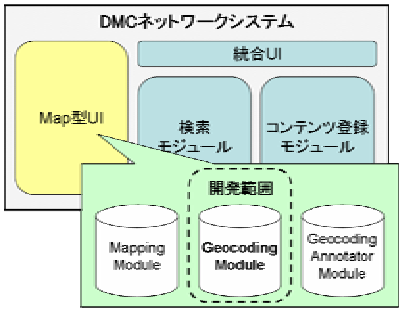


図 1

目次

Module 構造

Geocoding Module が正常に動作するために地理情報キャッシュテーブルと設定ファイルの二つのコンポーネントが必要となる。下の図 2 に Geocoding Module と各コンポーネントとの関連性を示す。

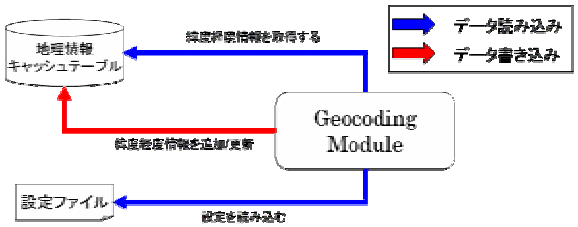


図 2

用語一覧

Map 型 UI：　DMC ネットワークシステムの新機能。（詳細は「DMC ネットワークシステム Map 型 UI 表示機能　機能仕様書」を参照）
Geocoding Module：　Map 型 UI の一部。渡された地名情報を処理し、それに該当する緯度経度情報を返す。
地理情報キャッシュテーブル：　緯度経度情報を蓄えるデータベース。以降「キャッシュテーブル」と表記。
Geocoding API：　外部に提供されている、緯度経度算出サービス。地理情報キャッシュテーブルに存在しない緯度経度情報を検索するために使用する。今回は Google 社のものを使用する（http://www.geocoding.jp/api）。
同名異地点：　「DMC ネットワークシステム Map 型 UI 表示機能　機能仕様書－用語一覧」を参照
設定ファイル：　Geocoding Module の動作を制御する設定ファイル。
プログラムログ：　Geocoding Module の動作履歴が記憶されているログファイル。
リクエストログ：　Geocoding Module に渡されたリクエスト情報が記憶されているファイル。

入出力

Gecoding Module はソケット通信を通してリクエストを受け取る。そして、レスポンスもソケット通信を通して返す。

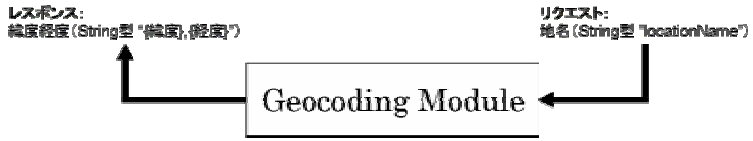


図 3

緯度経度情報は二つの数値（緯度と経度）のペアである。リクエスト内容によって、異なった地点の緯度経度情報を複数返すことがある。また、場合によってリクエストに該当する緯度経度情報がない場合もある。それぞれのケースのレスポンス形式を下記の表に記す。

緯度経度情報の数	レスポンス形式
1 個	String 型（“緯度,経度”） ※　緯度経度情報をコンマ区切りで返す
n 個	String 型（“緯度 1,経度 1;緯度 2,経度 2;緯度 3,経度 3:…緯度 n,経度 n”） ※　セミコロン区切りで異なった地点の緯度経度情報を分ける
0 個	null ※　リクエストに該当する緯度経度情報が見つからなかった場合、レスポンスとして null を返す

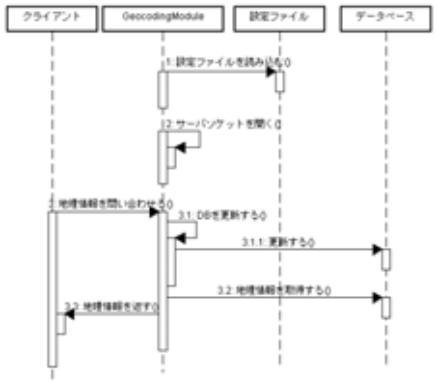
プログラム処理概要

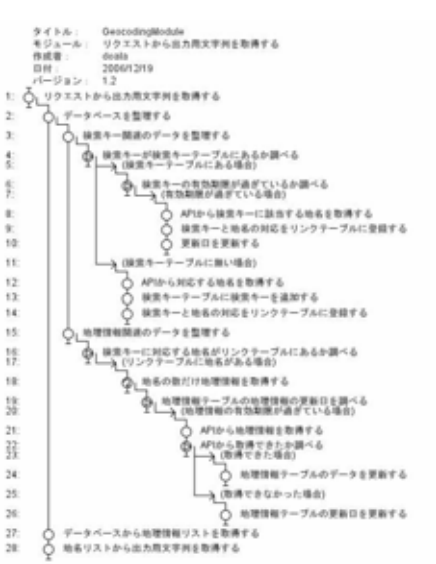
Geocoding Module のプログラム処理の概要を下記の HCP チャートで示す。

シーケンス図

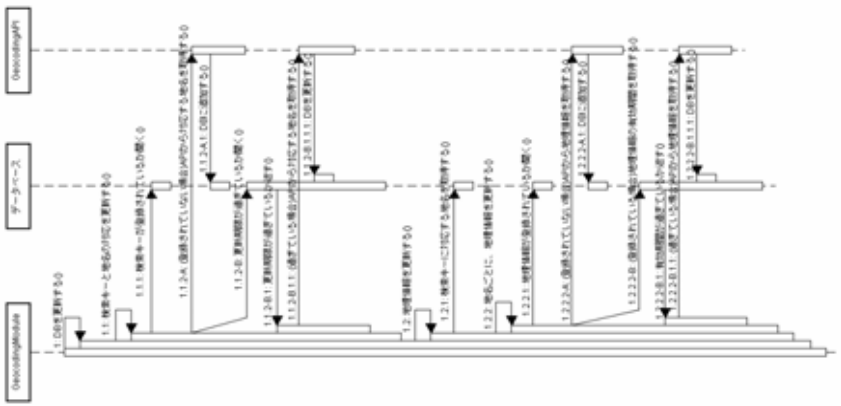
Geocoding Module の動作を下記のシーケンス図で示す。

・シーケンス図（概略）



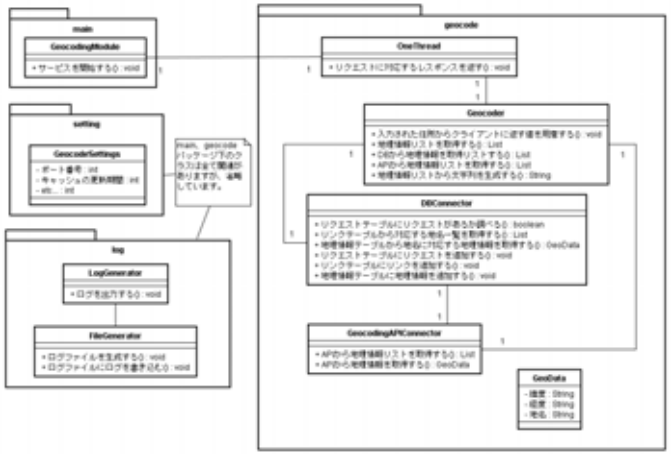


・ シーケンス図（DB を更新する）



クラス図

本モジュールにおけるクラス図は以下のようになっている。



地理情報キャッシュテーブル

地理情報キャッシュテーブルに Geocoding Module が扱う緯度経度情報をすべて格納する。Geocoding API で検索した地理情報もここに登録する。キャッシュの期限を設けることによって、定期的にデータベースの中身を更新する。具体的なデータベース設計は下記に示す。

使用するデータベース：　MySQL 8.15
文字エンコード：　UTF-8

テーブル名：　geocodingRequest
テーブル設計：

No.	列英名	列名	型	桁数
1	searchQuery	検索用語	char	100
2	requestTimes	リクエスト回数	int	-
3	lastUpdatedRequest	最終更新日	Date	-

テーブル名：　geocodingLink
テーブル設計：

No.	列英名	列名	型	桁数
1	searchQuery	検索用語	char	100
2	locationName	リクエスト回数	int	-
3	rank	優先順位	int	-

テーブル名：　geocodingData
テーブル設計：

No.	列英名	列名	型	桁数
1	locationName	位置	Char	100
2	latitude	緯度	Char	100
3	longitude	経度	Char	100
4	insertDate	登録日時	Date	-
5	lastUpdated	更新日時	Date	-

データ挿入例：

テーブル：geocodingRequest

searchQuery	locationName	lastUpdatedRequest
東京	4	2007-01-15
三田	6	2007-01-15
江戸	2	1970-01-01

テーブル：geocodingLink

searchQuery	requestTimes	rank
東京	東京	1
三田	三田駅（東京）	1

systemLog=on	システム関連の情報ログの出力可否（エラー関連は必ず出力される）
systemLogPath=./systemLog/	システム関連のログの出力先
accessLog=on	アクセスログの出力可否
accessLogPath=./accessLog/	アクセスログの出力先

・データベース関連の設定項目

項目名	説明
DBURL=jdbc:mysql:///GeocodingModuleDB?useUnicode=true&characterEncoding=UTF8	データベースの URL
DBUserName=root	データベースのユーザ名
DBPass=pass	データベースユーザのパス

三田	三田駅（東京）	2
江戸	江戸	1

テーブル：geocodingData

locationName	latitude	longitude	insertDate	lastUpdated
東京	100	200	2007-01-15	2007-01-15
三田駅（東京）	150	200	2007-01-15	2007-01-15
三田駅（兵庫）	200	300	2007-01-15	2007-01-15
江戸	100	200	1970-01-01	1970-01-01

※　同名異位置点の緯度経度情報は該当性順位を付けて、それぞれ別レコードとしてデータベースに登録される。

※　検索用語に該当する緯度経度情報が Geocoding API を用いても見つからなかった場合、null 値としてデータベースに登録される。

システムログ

Geocoding Module の動作をシステムログとして出力する。

ログのレベルにはエラー（致命的なエラーが発生したことを知らせる）、警告（致命的ではないが意図通り動かない事態が発生したことを知らせる）、情報（起動や設定ファイル読み込みといった情報を知らせる）の 3 レベルがあり、設定ファイルによって情報レベルを出力するかしないかを決定できる。

具体的なシステムログ仕様を下記に示す。

ファイル名：　programLog_[yyyymmdd].txt
ファイル形式：　txt
文字エンコード：　UTF-8

システムログ例 (# の行は実際には出力されない)：

```
# Geocoding Module 開始
Jan 01 00:00:00 [INFO] geocoding module start

# 設定ファイルの読み込み
Jan 01 00:00:00 [INFO] reading config file (geoConfig.txt)

# 設定ファイルの読み込みに失敗したとき
Jan 01 00:00:00 [ERROR] failed reading config file (geoConfig.prop)
Jan 01 00:00:00 [ERROR] ERROR config doesn't exist
```


設定ファイル

Geocoding Module の動作を制御する設定ファイルは別ファイルとして用意する。下記に具体的な仕様を示す。

設定ファイル名：	geoConfig.prop
ファイル形式：	prop
文字エンコード：	UTF-8

設定ファイル仕様：

・全般

項目名	説明
modulePort=8080	Geocoding Module が使用するポート番号
requestEncode=UTF-8	入出力に使う文字列のエンコード
requestWaitTime=5000	API 接続のインターバル時間（ミリ秒単位）
returnMax=2	Geocoding Module が返す結果の数
cacheExpiration=15	地理情報の更新期間（日単位）
searchKeyExpiration=15	検索キーの更新期間（日単位）
apiURL=http://www.geocoding.jp/api/?q=	Web 上の地理情報取得サービスの URL

・ログ関連の設定項目

項目名	説明
logEncode=UTF-8	ログのエンコード

エラー処理概要

Geocoding Module の動作中にエラーが発生した場合、以下のような対処をとる。
その際、エラーメッセージをシステムログに出力する。

エラーNo.	エラータイプ	エラーメッセージ	エラー内容	エラー処理
10	ERROR	Config file doesn't exist	設定ファイルが見つからない	プログラム終了
11	ERROR	Config file broken or has been modified	設定ファイルが壊れている	プログラム終了
20	ERROR	Failed creating server socket	サーバソケットの作成に失敗	プログラム終了
30	WARNING	Bad request	不正なリクエストを受け取った	レスポンスとしてnullを返す
40	WARNING	Cannot create program log	プログラムログの作成に失敗	
41	WARNING	Cannot write to program log	プログラムログの書き込みに失敗	
42	WARNING	Cannot create request log	リクエストログの作成に失敗	
43	WARNING	Cannot write to request log	リクエストログの書き込みに失敗	
50	WARNING	Cannot connect to API	Geocoding API 接続に失敗	レスポンスとしてnullを返す
51	ERROR	Cannot connect to database	キャッシュテーブルの接続に失敗	レスポンスとしてnullを返す
60	ERROR	Cannot read from input stream	入力ストリームから読み込めない	レスポンスとしてnullを返す
61	ERROR	Cannot write to output stream	出力ストリームに書き込めない	


```
Jan 01 00:00:00 [ERROR] ERROR config file is broken or has been modified

# サーバソケットの作成
Jan 01 00:00:00 [INFO] creating server socket at port 8080

# サーバソケットの作成に失敗したとき
Jan 01 00:00:00 [ERROR]ERROR failed creating server socket

# サービスの開始を知らせる
Jan 01 00:00:00 [INFO]starting service

# データベースへ接続する
Jan 01 00:00:00 [INFO]connecting to database

# データベースに接続出来なかったとき
Jan 01 00:00:00 [ERROR]ERROR database connection failed

# Geocoding Module 終了
Jan 01 00:00:00 [INFO]geocoding module terminated
```

※　実際に「#」のコメント行は出力されません

Geocoding Module テスト設計書

2006 年 11 月 23 日作成
慶應義塾大学 大岩研究会 g-mod チーム

アクセスログ

Geocoding Module が受けたリクエストの内容をログファイルとして出力することが出来る。出力の可否は設定ファイルの中で設定することが出来る。
具体的なリクエストログ仕様を下記に示す。

ファイル名：　requestLog_[yyyymmdd].txt
ファイル形式：　txt
文字エンコード：　UTF-8

アクセスログ例：(# の行は実際には出力されない)：

```
# リクエストを受け取ったとき
Jan 01 00:00:00 [INFO] received request from hostA [192.168.0.0]
Jan 01 00:00:00 [INFO] request "新東京"

# リクエストに該当する緯度経度情報が見つかったとき
Jan 01 00:00:00 [INFO] found record for "新東京"

# リクエストに該当する緯度経度情報が見つからなかった時
Jan 01 00:00:00 [INFO] could not find any lat/long entries for "新東京"
```

I.　単体テスト

今回 g-mod が作成した緯度経度算出システムがクライアントの要求する機能が満たされているかを，緯度経度算出システム単体でのテストをする．

版	作成日	作成者	承認者	更新箇所
0.1	2006/12/2	松田　航	－	－
0.2	2006/12/10	松田　航		
1.0	2007/01/15	佐藤俊之		細部修正
1.1	2007/01/23	佐藤俊之		松澤さんレビューに伴う処理の流れの変更によるテストの変化

I-1.　単体テスト対応表

単体テストにおいて試験する仕様と，それに対応するテストは以下のようになっている．
仕様の流れについては設計書に記述したものを参照．

仕様分類	仕様名	正常処理テスト番号	異常処理テスト番号
起動	サービスを立ち上げる	1.1.1	2.1.1
設定	設定ファイルを読み込む	1.2.1, 1.2.2	2.2.1, 2.2.2
ソケット通信	サーバソケットを作成する	1.3.1	2.3.1
本処理	リクエストから出力用文字列を取得する	1.4.1, 1.5.1	2.4.1, 2.5.1
	DB を更新する	1.5.2.1	2.5.2, 2.5.3
	DB から地理情報を取得する	1.5.2.2	2.5.2
DB 関連	検索キーが DB にあるか調べる	1.5.3.1.1	2.5.2
	検索キーの期限が切れているか調べる	1.5.3.1.2	2.5.2
	検索キーを更新する	1.5.3.1.3	2.5.2
	検索キーを新規に追加する	1.5.3.1.4	2.5.2
	検索キーの検索回数を増やす	1.5.3.1.5	2.5.2
	検索キーに対応する地名リストを取得する	1.5.3.2.1	2.5.2
	地名に対応する地理情報が DB にあるか調べる	1.5.3.2.2	2.5.2
	地理情報の有効期限を調べる	1.5.3.2.3	2.5.2
	地理情報を更新する	1.5.3.2.4	2.5.2
	地理情報を新規に追加する	1.5.3.2.5	2.5.2
	検索キーに対応する地理情報を全て取得する	1.5.3.3.1	2.5.2
API 関連	API から検索キーに対応する地名一覧を取得する	1.5.4.1	2.5.3

目次

		1.5.3.2.4	地理情報を更新できるか
		1.5.3.2.5	地理情報を追加できるか
1.5.3.3	（全般）	1.5.3.3.1	検索キーに対応する地理情報を取得できるか
1.5.4	API 関連	1.5.4.1	検索キーに対応する地名一覧が取得できるか
		1.5.4.2	地名に対応する地理情報を取得できるか
1.6	ログ関連	1.6.1	ログのパスが正しいか判定できるか
		1.6.2	ログが出力できるか

2.異常動作

システムが正常に動作していない際に，仕様通りの動作をしているかをテストする．

項目は以下のようになっている．

分類番号	分類	テスト番号	テスト内容
2.1	起動	2.1.1	起動できなかった場合，プログラムログにエラーを出力するか．
2.2	設定ファイル	2.2.1	設定ファイルが存在しない場合，プログラムログにエラーを出力し，プログラムを中止するか
		2.2.2	設定ファイルからデータが取得できなかった場合，プログラムログにエラーを出力し，プログラムを中止するか
2.3	ソケット通信	2.3.1	ソケット通信の確保できなかった場合，プログラムログにエラーを出力するか
2.4	入力	2.4.1	引き渡されたデータが扱えないものの場合，リクエストログに出力し，空データを返すか
2.5	処理	2.5.1	引き渡されたデータごとに地理情報を取得できなかった場合，リクエストログに出力し、空データを返すか
		2.5.2	DB にアクセスできない場合，プログラムログに出力し，空データを返すか
		2.5.3	API にアクセスできない場合，プログラムログに出力し，空データを返すか

	API から地理情報を取得する	1.5.4.2	2.5.3

I -1. テスト仕様詳細

別資料を参照

I -2. テスト仕様一覧

単体テストの仕様一覧は以下のようになっている．

1.正常動作

システムが正常に動作している際に，仕様通りの動作をしているかをテストする．

項目は以下のようになっている．

分類番号	分類	テスト番号	テスト内容
1.1	起動	1.1.1	正常にサービスが立ち上がるか
1.2	設定ファイル	1.2.1	Property ファイルが正しい形式で存在する場合，読みこめているか
		1.2.2	property ファイルから正しいデータが取得できているか確認するか
1.3	ソケット通信	1.3.1	ソケット通信の確保ができるか
1.4	入力	1.4.1	引き渡されたデータを扱うことができるか
1.5	地理情報取得	1.5.1	リクエストを受け、それに対応する文字列が作成できるか
1.5.2	処理詳細	1.5.2.1	データベースを正しく更新するか
		1.5.2.2	データベースから地理情報が取得できるか
1.5.3.1	DB(検索キー)	1.5.3.1.1	検索キーが検索キーテーブルにあるか取得できるか
		1.5.3.1.2	検索回数を更新できるか
		1.5.3.1.3	検索キーの有効期限が切れているかどうか判定できるか
		1.5.3.1.4	検索キーを更新できるか
		1.5.3.1.5	検索キーを追加できるか
1.5.3.2	（地理情報）	1.5.3.2.1	検索キーに対応する地名一覧が取得できるか
		1.5.3.2.2	地名に対応する地理情報が地理情報テーブルに存在するか
		1.5.3.2.3	地理情報の有効期限が切れているかどうか判定できるか

		2.3	入力異常	入力データの形式が異常の場合，空情報が返ったことを確認する
		2.4	DB 異常	DB の形式が異常の場合，処理を中止する，プログラムを終了する
				空情報が返ったことを確認する

結合テスト

今回 g-mod が作成した緯度経度算出システムが,DMC システムと接続した際に,両システムが共に正常な稼働する事ができるかをテストする.
テストは運用環境下にあるシステムを対象に行う.

Ⅱ-1. テストデータ定義

テストに用いるデータは以下のものを用いる

1. property ファイル
   (ア)正規フロー用
   (イ)異常フロー（シナリオ番号 2.2）
2. 入力ファイル
   (ア)正規フロー
   (イ)異常フロー（シナリオ番号 2.3）

Ⅱ-1. テストシナリオ

テストシナリオは以下のようになっている.

分類番号	分類	シナリオ番号	シナリオ名	テスト内容
1	正常フロー	1.1	初回起動	DB に何も入っていない状態で，リクエストする
				DB に入力があることを確認する
				ファイルが出力されたことを確認する
		1.2	次回起動	DB にデータが入っている状態で，新規情報をリクエストする
				DB に新規データ分の入力があることを確認する
				DB に新規データ分以外は更新されていないことを確認する
				既存情報をリクエストする
				DB の最終リクエスト日時以外は更新されていないことを確認する
				ファイルが出力されたことを確認する
		1.3	地理情報更新	地理情報テーブルに更新期間の切れたデータがあり，それに対応するリクエストをする
				DB においてそのデータが更新されていることを確認する
				ファイルが出力されたことを確認する
		1.4	リクエスト更新	リクエストテーブルに更新期間の切れたデータがあり，それに対応するリクエストをする
				DB においてそのデータが更新されていることを確認する
				ファイルが出力されたことを確認する
2	異常フロー	2.1	設定異常 1	設定ファイルが存在しなかった場合，処理を中止し，プログラムを終了する
				DB への入力及び出力ファイルに変化が無いことを確認する
		2.2	設定異常 2	設定ファイルが異常な形式の場合，処理を中止し，プログラムを終了する
				DB への入力及び出力ファイルに変化が無いことを確認する

1	扱うことが出来る場合、TRUEを返す	"東京"	T
2	扱うことが出来ない場合、FALSEを返す	null	F

テスト番号	1.5.1		
テスト名称	リクエストを受け、それに対応する文字列が作成できるか		
入力形式	String(検索キー)		
出力形式	List<地理情報データ>		
テストケース番号	目的	入力	出力(Listの中
1	地理情報が1つ取得できる場合、指定した形式に沿って返す	"東京"	"100,200"
2	地理情報が複数取得できる場合、指定した形式に沿って返す	"三田"	"100,200;200,3
3	取得できない場合、nullを返す	"あああああ"	null

テスト番号	1.5.2.1		
テスト名称	データベースを正しく更新するか		
入力形式	String(検索キー)		
出力形式	DBが更新される		
テストケース番号	目的	入力	出力(DBの中
1	新規に検索したキーの場合、検索キーが追加される	"横浜"	
2	検索キーの更新期間が過ぎている場合、検索キーを更新する	"江戸"	
3	更新期間が過ぎていない場合、検索回数のみを更新する	"東京"	
4	地理情報がDBに無い場合、地理情報を取得し追加する	"横浜"	
5	地理情報の更新期間が過ぎている場合、地理情報を更新す	"江戸"	
6	地理情報の更新日が過ぎていない場合、更新をしない	"東京"	

テスト番号	1.5.2.2		
テスト名称	データベースから地理情報が取得できるか		
入力形式	String(検索キー		
出力形式	List<GeoData>		
テストケース番号	目的	入力	出力
1	地理情報が1つ取得できる場合、1つ取得できる	"東京"	"東京"の地理
2	地理情報が複数取得できる場合、複数取得できる	"三田"	"三田(東京)"
3	地理情報が取得できない場合、空のリストが取得できる	"あああああ"	何も取得でき

テスト番号	1.5.3.1.1
テスト名称	検索キーが検索キーテーブルにあるか取得できるか
入力形式	String(検索キー),回数

出力形式	boolean		
テストケース番号	目的	入力	出力(回数)
1	キーがデータベースに存在する場合、Trueを返す	"東京"	T
2	キーがデータベースに存在しない場合、Falseを返す	"横浜"	F

テスト番号	1.5.3.1.2		
テスト名称	検索回数を更新できるか		
入力形式	String(検索キー)		
出力形式	DBの検索回数の値が変わる		
テストケース番号	目的	入力	出力
1	検索キーの検索回数を増やし、更新する	"東京"	東京のreques

テスト番号	1.5.3.1.3		
テスト名称	検索キーの有効期限が切れているかどうか判定できるか		
入力形式	String(検索キー)		
出力形式	boolean		
テストケース番号	目的	入力	出力(リンク値
1	検索キーの期限が切れている場合、Trueを返す	"東京"	T
2	検索キーの期限が切れていない場合、Falseを返す	"江戸"	F

テスト番号	1.5.3.1.4		
テスト名称	検索キーを更新できるか		
入力形式	String(検索キー)、List<String>(地名)		
出力形式	DBが更新されている		
テストケース番号	目的	入力	出力(String[
1	検索キーの更新日と、リンクテーブルを更新する	"東京"、List("東京")	更新日とリン

テスト番号	1.5.3.1.5		
テスト名称	検索キーを追加できるか		
入力形式	String(検索キー)、List<String>(地名)		
出力形式	DBか更新されている		
テストケース番号	目的	入力	出力(Listの中
1	地名がある場合、検索キーテーブルとリンクテーブルに追加	"東京"、List("東京")	検索キー・リン
2	地名が無い場合、検索キーテーブルのみを更新する	"あああああ"、NULL	検索キーテー

テスト番号	1.5.3.2.1

テストケース詳細仕様書

テスト番号	1.1.1		
テスト名称	正常にサービスが立ち上がるか		
入力形式	無し		
出力形式	サービスが立ち上がる		
テストケース番号	目的	入力	出力
1	サービスを立ち上げる	無し	サービスが立

テスト番号	1.2.1		
テスト名称	Propertyファイルが正しい形式で存在する場合，読みこめているか		
入力形式	無し		
出力形式	boolean		
テストケース番号	目的	入力	出力
1	ファイルが存在する場合、読み込めることを返す	無し	T
2	ファイルが存在しない場合、読み込めないことを返す	無し	F
3	項目が欠けた状態で存在する場合、読み込めないことを返す	無し	F

テスト番号	1.2.2		
テスト名称	propertyファイルから正しいデータが取得できているか確認するか		
入力形式	無し		
出力形式	boolean		
テストケース番号	目的	入力	出力
1	読み込んだ値が正常な場合、読み込めることを返す	無し	T
2	読み込んだ値が正常でない場合、読み込めないことを返す	無し	F

テスト番号	1.3.1		
テスト名称	ソケット通信の確保ができるか		
入力形式	無し		
出力形式	ソケットが立ち上がる		
テストケース番号	目的	入力	出力
1	ソケットを立ち上げる	無し	ソケットが立ち

テスト番号	1.4.1		
テスト名称	引き渡されたデータを扱うことができるか		
入力形式	String		
出力形式	boolean		
テストケース番号	目的	入力	出力

1	ログが出力可能なパスの場合、Trueを返す	./Log/	T
2	ログが出力不能なパスの場合、Falseを返す	¥¥"¥¥¥"	F/

テスト番号	1.6.2		
テスト名称	ログが出力できるか		
入力形式	無し		
出力形式	ログファイル		
テストケース番号	目的	入力	出力
1	ログファイルが存在しない場合、新規に作成する	"起動"	ログが出力さ
2	ログファイルが存在する場合、ファイル内に書き込む	"起動"	既存のファイ

テスト番号	1.5.3.2.1		
テスト名称	検索キーに対応する地名一覧が取得できるか		
入力形式	String(検索キー)		
出力形式	List<地名>		
テストケース番号	目的	入力	出力
1	検索キーに対応する地名が1つの場合、1つだけ返す	"東京"	List("東京")
2	検索キーに対応する地名が複数ある場合、全て返す	"三田"	List("三田(東
3	検索キーに対応する地名が無い場合、NULLを返す	"あああああ"	List(null)

テスト番号	1.5.3.2.2		
テスト名称	地名に対応する地理情報が地理情報テーブルに存在するか返す		
入力形式	String(地名)		
出力形式	boolean		
テストケース番号	目的	入力	出力(DBより取
1	地理情報テーブルに存在する場合、Trueを返す	"三田"	T
2	地理情報テーブルに存在しない場合、Falseを返す	"あああああ"	F/

テスト番号	1.5.3.2.3		
テスト名称	地理情報の有効期限が切れているかどうか判定できるか		
入力形式	String(地名)		
出力形式	boolean		
テストケース番号	目的	入力	出力
1	期限が切れている場合、Trueを返す	"江戸"	T
2	期限が切れていない場合、Falseを返す	"東京"	F/

テスト番号	1.5.3.2.4		
テスト名称	地理情報を更新できるか		
入力形式	地理情報データ		
出力形式	地理情報テーブルが更新される		
テストケース番号	目的	入力	出力(DBの最
1	地理情報が更新される	"江戸"、100、50	入力通りの地

テスト番号	1.5.3.2.5
テスト名称	地理情報を追加できるか
入力形式	String(地名)
出力形式	地理情報テーブルが更新される

テストケース番号	目的	入力	出力
1	地理情報を追加する	"カナダ"、50、20	入力通りの地

テスト番号	1.5.3.3.1		
テスト名称	検索キーに対応する地理情報を取得できるか		
入力形式	String(地名)		
出力形式	List<地理情報データ>		
テストケース番号	目的	入力	出力(Listの中
1	地理情報が1つ存在する場合、1つ入ったリストを返す	"東京"	東京
2	地理情報が複数存在する場合、複数入ったリストを返す	"三田"	三田(東京)、
3	該当するものが無い場合、1つも入っていないリストを返す	"あああああ"	null

テスト番号	1.5.4.1		
テスト名称	検索キーに対応する地名一覧が取得できるか		
入力形式	String(地名)		
出力形式	List<String>		
テストケース番号	目的	入力	出力(リストの
1	キーに対応する地名が1つある場合、地理情報を保存し、1つ	"東京"	"東京"
2	キーに対応する地名が複数ある場合、全て返す	"三田"	"三田(東京)"
3	対応する地名が無い場合、nullを返す	"あああああ"	null

テスト番号	1.5.4.2		
テスト名称	地名に対応する地理情報を取得できるか		
入力形式	String(地名)		
出力形式	List<地理情報データ>		
テストケース番号	目的	入力(Listの中)	出力
1	保存してある場合、保存されている所から取得する	東京	"100,200"
2	保存されていない場合、保存	三田(東京)、三田(兵庫	"100,200;200,
3	存在しない場合、nullを出力する	null	null

テスト番号	1.6.1
テスト名称	ログのパスが正しいか判定できるか
入力形式	String(ログのパス)
出力形式	boolean
テストケース番号	目的	入力	出力

″）

録される

の地理情報

録される

テスト結果報告書

・形式について
テストは複数回に分けて行った。
各自が担当箇所のテストプログラムを作成し、それを基にコーディングを行った。
1月8日に各自の担当箇所のマージ作業を行い、その際に全体で行った単体テストを第一回目の記録とした。
それ以前に各自の担当箇所で出たバグは計測していない。
その後、マージ作業によって生まれた不具合の修正や、及びその後の設計変更に伴うテストの変化があり、
最終的に1月24日に2回目の単体テストを行った。
既にマージしてある状態であり、各自のパーツの繋がりができているため、
ここで発見されたエラーは1つも無かった。

テスト結果一覧

テスト番号	テストケース番号	1回目（1月8日）	2回目（1月24日）
1.1.1	1	○	○
1.2.1	1	×	○
	2	×	○
	3	×	○
1.2.2	1	○	○
	2	○	○
1.3.1	1	○	○
1.4.1	1	○	○
	2	○	○
1.5.1	1	○	○
	2	○	○
	3	×	○
1.5.2.1	1	○	○
	2	×	○
	3	×	○
	4	○	○
	5	×	○
	6	×	○
1.5.2.2	1	○	○
	2	○	○
	3	○	○
1.5.3.1.1	1	○	○
	2	○	○
1.5.3.1.2	1	○	○
1.5.3.1.3	1	○	○
	2	○	○
1.5.3.1.4	1	×	○
1.5.3.1.5	1	○	○
	2	○	○
1.5.3.2.1	1	○	○
	2	○	○
	3	○	○
1.5.3.2.2	1	○	○
	2	○	○
1.5.3.2.3	1	○	○
	2	○	○
1.5.3.2.4	1	×	○
1.5.3.2.5	1	○	○
1.5.3.3.1	1	○	○
	2	○	○
	3	○	○
1.5.4.1	1	×	○
	2	○	○
	3	○	○
1.5.4.2	1	○	○
	2	×	○
	3	○	○
1.6.1	1	×	○
	2	×	○
1.6.2	1	×	○
	2	×	○
2.1.1	1	○	○
2.2.1	1	○	○
2.2.2	1	○	○
2.3.1	1	○	○
2.4.1	1	○	○
2.5.1	1	○	○
2.5.2	1	○	○
2.5.3	1	○	○
		42/58	58/58

Geocoding Module　インストールマニュアル

2007 年 1 月 17 日作成
慶應義塾大学　大岩研究会 g-mod チーム

版	作成日	作成者	承認者	更新箇所
0.1	2007/01/17	平澤秀幸	ー	ドキュメント作成

目次

推奨環境

CPU	300MHz 以上
メモリ	128 RAM 以上
ハードディスク	1GB 以上の空き容量
その他	JRE 1.5 以上がインストール済み MySQL 5.0 以上がインストール済み

インストール手順

ファイル一覧

Geocoding Module の動作に必要なファイルを下記の表で記す。

ファイル名	**説明**
GeocodingModule.jar	地名から緯度経度情報を割り出すモジュールの本体
geoConfig.prop	Geocoding Module 用の設定ファイル GeocodingModule.jar と同じディレクトリにおく必要がある
Lib フォルダ	MySQL ドライバであるmysql-connector-java-5.0.4-bin.jarが格納されているフォルダ
GeocodingModule.sh	GeocodingModule を呼び出すスクリプトファイル
Data2.sql	DB の初期データSQL ファイル

Geocoding Module　インストール・操作マニュアル

2007 年 1 月 17 日作成
慶應義塾大学 大岩研究会 g-mod チーム

版	作成日	作成者	承認者	更新箇所
0.1	2007/01/17	平澤秀幸	―	ドキュメント作成

モジュールインストール方法

Zip ファイルを解凍すると以下のような配置でファイルが出力されます。

```
GeocodingModule.jar geoConfig.prop GeocodingModule.sh lib data2.sql
```

上記のファイルを任意のディレクトリを配置する。

データベース作成

地理情報を保存するためのデーベースを作成する必要があります。作成方法は以下のようになります。

```
% mvsql -u root -p      //root でログイン
% Enter password:       //password を入力する（初回インストール時には無し）

mysql> CREATE DATABASE geocodingmoduledb;          //DB 作成する。DB 名は任意
```


初期データ挿入

初期データのあるディレクトリで以下のコマンドを使いデータベースに挿入します。

```
% mvsql [DB 名] < [SQL ファイル] -u [DB ユーザ名] -p [DB パスワード]
```

例

```
% mvsql geocodingmoduledb < data2.sql -u root -p
```

PATH設定

GeocodingModule を起動するための PATH の設定を行います。
/etc/profile に以下のラインを追加します

```
export G_HOME=[GeocodingModuke.sh の置いてある任意の PATH 名]
export PATH="$PATH:$G_HOME":
例
export G_HOME=/home/g-mod/GeocodingModule
export PATH="$PATH:$G_HOME":
```

GeocodingModule.sh の権限を変更する必要があります。

```
% chmod 777 GeocodingModule.sh
```

以上でインストールは完了です。

目次

インストール手順

ファイル一覧

Geocoding Module の動作に必要なファイルを下記の表で記す。

ファイル名	説明
GeocodingModule.jar	地名から緯度経度情報を割り出すモジュールの本体
geoConfig.prop	Geocoding Module 用の設定ファイル GeocodingModule.jar と同じディレクトリにおく必要がある

インストール方法

GeocodingModule.jar と geoConfig.prop の二つのファイルを Unix の mv コマンドなどを用いて任意のディレクトリに設置する。

```
% mv GeocodingModule.jar geoConfig.prop newDirectory
% cd newDirectory
% ls
GeocodingModule.jar geoConfig.prop
```


操作手順

起動手順

Gecoding Module は java –jar GeocodingModule.jar コマンドで起動する。無事起動が終了したら下記のようなダイアログがコンソールに表示される。

```
% java -jar GeocodingModule.jar
Geocoding Module start
Reading config file
starting service
creating server socket
Server is ready at port:8080
```

終了方法

Geocoding Module を終了するためには Geocoding Module のプロセスを Unix の kill コマンドで終了、もしくはモジュールが起動しているコンソールで Ctrl+C と打つ。

方法 1

```
% kill [process ID]
```


方法 2

```
% java -jar GeocodingModule.jar
Geocoding Module start
Reading config file
starting service
creating server socket
Server is ready at port:8080
…
…
…
[Ctrl+C]
%
```

報告日　2006年10月12日

週 間 報 告 書

プロジェクト名　g-mod
報告者：　谷田部　学

報告期間	2006年10月5日 ～ 2006年10月12日	
実績	No.	内容	状態
	1	人材募集資料作成	完了
	2	チーム決め	完了
	3	チームメンバーへ要件の説明	完了
	4	WBS作成	完了
	5	プロジェクト定義書作成	完了
	6	週間報告書作成	完了
	7	進捗報告資料作成	完了
	8	要望ヒアリングの為の日程調整	完了
課題・対策	1. 要件定義の承認者	
	10月16日～25日まで島津先生が不在となる。	
	その為、要件定義の承認をしてくれる代替の方を決めて頂く必要あり。	
	→不在期間中はDMCの斉藤さんを含む他の方々で質問に対応頂ける。	
	2. 開発環境について	
	DMCシステム検証機を使用することが可能。	
	sshかtelnetでのリモート接続可能だが、ftp等のファイル転送が行えない可能性がある。	
	→校内に開発環境を構築する可能性あり。	
今週予定	1. 要件定義	
	10月12日　島津先生から要件のヒアリング　(作業者:佐藤、平澤、松田、谷田部)	
	10月12日～10月18日　要件定義書の作成　(作業者:佐藤、平澤、松田、谷田部)	
	10月18日　要件定義書レビュー(作業者:佐藤、谷田部)	
	10月19日　要件定義書　承認	
	2. Wikiにあるプロジェクト用ページの作成	
	Wikiを使用して情報整理、共有が行えるようにページを編集する	

作業時間サマリ

No.	ジョブ名	時間(h)
1	資料作成	1.0
2	ML対応	1.0
3	フィールドワーク	1.0
4	授業	1.0
5	気にした時間	1.0
6	その他	0.0
	合計	5.0

備考

週 間 報 告 書

プロジェクト名　g-mod
報告者：　谷田部　学

報告期間	2006年10月12日 ～ 2006年10月18日		
実績	No.	内容	担当	状態(完了・継続・未着手)
	1	プロジェクト定義書作成	谷田部	完了
	2	スケジュール再作成	谷田部	完了
	3	週間報告書作成	谷田部	完了
	4	要件定義書目次作成	佐藤	継続
	5	要件定義書作成	全員	未着手
	6	進捗報告資料作成	佐藤	完了
	7	進捗報告資料の精査	全員	完了
	8	クライアント要望のヒアリング	全員	完了
	9	ヒアリングログの作成	佐藤	完了
	10	Wikiの整理	平澤	継続
	11	プロジェクト目標設定	全員	完了
課題・対策	1. メンバー間の認識のズレ		
	スケジュールとPM指示がハッキリしていなかった為にメンバー間で認識のズレが発生していた。		
	PMからスケジュールの説明を実施する。		
	指示は形に残す。		
	2. クライアントからの資料送付遅延		
	クライアント都合により、頂く予定の資料がまだ頂けていない状態になっている。		
	把握している範囲で作業を進める。		
今週予定	1. 要件不明点の洗い出し		
	要件の不明点を洗い出して、クライアントに質問を投げる。		
	2. 要件定義書作成		
	目次の精査、要件定義書フォーマット作成、要件定義書作成を実施する。		
	不明な箇所以外で手を付けられる部分を作成する。		
	3. 要件定義書承認		
	要件定義書に承認を頂く。		

作業時間サマリ

No.	ジョブ名	時間(h) 谷田部	佐藤	平澤	松田	合計	
1	資料作成	3.0	5.0	0.5	0.0	8.5	5.5
2	ML対応	1.0	0.5	1.0	1.5	4.0	3.0
3	フィールドワーク	0.0	0.0	0.0	0.0	0.0	0.0
4	授業	7.0	3.0	3.0	3.0	16.0	9.0
5	気にした時間	15.0	1.5	1.0	1.0	18.5	3.5
6	その他	0.0	0.0	0.0	0.0	0.0	0.0
	合計	26.0	10.0	5.5	5.5	47.0	21.0

所感

自社の業務が忙しくなってしまい、g-modの業務が滞ってしまった。
なるべく時間を決めてg-mod業務を行うようにしているのだが、割り込み作業が入ってしまう状態が続いている。
メンバーとはメールでやり取りをしているが、やはり実際に会ってマメにミーティングを行える方がいいと痛感している。
要件やスケジュールに対する認識の違いも発生しており、原因は私の指示があいまいであった事にある。
今後は、できるだけ的確な指示をすると共にコミュニケーションツールを使って形に残しておくようにする。

週 間 報 告 書

プロジェクト名　g-mod
報告者：　谷田部　学

報告期間	2006年10月19日 ～ 2006年10月25日		
実績	No.	内容	担当	状態(完了・継続・未着手)
	1	週間報告書作成	谷田部	完了
	2	要件定義書目次作成	佐藤	完了
	3	要件定義書作成	佐藤	完了
	4	進捗報告資料作成	平澤	完了
	5	進捗報告資料の精査	全員	未着手
	6	実現性の確認　(1)XMLの扱い	平澤	完了
	7	(2)GeoModAPI使用方法	平澤、佐藤	完了
	8	(3)DB接続	松田	完了
	9	(4)外部定義ファイル読み込み	松田	継続
	10	Q&Aの作成と送付	平澤	完了
	11	中間報告に向けての資料準備	谷田部	継続
課題・対策	1. コミュニケーションルールミス		
	コミュニケーションルールの通知がメンバーに行き渡っていなかった為、		
	嶋津先生から指摘あり。		
	→コミュニケーションルールの再通知を行う。		
	2. 共通認識のズレ		
	クライアントからヒアリングした内容やチームミーティングで話した内容を聞き逃しているメンバーがいる。		
	ログからも漏れていた項目もあったのでしょうがない部分もある。		
	ミーティング時に聞き逃していそうな人に確認する。		
今週予定	1. 要件定義書 ver.0.1を嶋津先生へ送付		
	恐らく修正個所があるので、11月2日までに修正した要件定義書を送付		
	要件定義書の承認目標は11月2日！		
	2. 嶋津先生にQ&Aの回答を頂く g-mod返信希望:10月31日		
	3. 詳細設計書の作成		
	4. テスト設計書作成		

作業時間サマリ

No.	ジョブ名	時間(h) 谷田部	佐藤	平澤	松田	合計	
1	資料作成	1.0	4.0	8.0	6.0	19.0	18.0
2	ML対応	4.0	3.0	3.0	2.0	12.0	8.0
3	フィールドワーク	0.0	0.0	0.0	0.0	0.0	0.0
4	授業	8.0	3.0	3.0	3.0	17.0	9.0
5	気にした時間	5.0	2.0	1.0	15.0	23.0	18.0
6	その他	0.0	0.0	3.0	1.0	4.0	4.0
	合計	18.0	12.0	18.0	27.0	75.0	57.0

所感

プロジェクトの進捗に遅れは出ているものの、実装フェーズで取り戻せる予定である。
今後はなるべくスケジュールに乗って仕事をこなしていきたいが、メンバーと会える時間が少ないので難しそうだ。
共通認識と意思の疎通が出来るができるような策を考えないと・・・Wikiをもっと有効に使えないか？

週 間 報 告 書

プロジェクト名　g-mod
報告者：　谷田部　学

報告期間	2006年10月26日 ～ 2006年11月1日		
実績	No.	内容	担当	状態(完了・継続・未着手)
	1	週間報告書作成	谷田部	完了
	2	プロジェクト提案書作成	佐藤	継続
	3	進捗報告資料作成	松田	完了
	4	進捗報告資料の精査	松田	完了
	5	詳細設計書作成	平澤	継続
	6	中間報告に向けての資料準備	谷田部	継続
	7			
	8	Q&A対応	平澤	継続
	9			
	10			
	11			
課題・対策	1. スケジュール遅延		
	プロジェクト提案書への承認が14日遅延している。		
	並行して作業を進めているため他タスクへの大きな影響は無い。		
	但し、要件が大きく変更された場合は		
	2. クライアントとの連絡スケジュール		
	クライアントとの連絡が取れなかった。		
	連絡が取れるようにクライアントとスケジュール調整を行う		
今週予定	1. プロジェクト提案書 ver.1.0に承認頂く		
	2. 中間発表資料　作成		
	3. 詳細設計書の作成		
	4. テスト設計書作成		

作業時間サマリ

No.	ジョブ名	時間(h) 谷田部	佐藤	平澤	松田	合計	
1	資料作成	1.5	3.0	6.0	5.0	15.5	14.0
2	ML対応	0.5	1.0	3.0	1.0	5.5	5.0
3	フィールドワーク	0.0	0.0	0.0	0.0	0.0	0.0
4	授業	8.0	3.0	3.0	3.0	17.0	9.0
5	気にした時間	0.5	3.0	0.0	0.0	3.5	3.0
6	その他	0.0	0.0	0.0	0.0	0.0	0.0
	合計	10.5	10.0	12.0	9.0	41.5	31.0

所感

今週はモチベーション上がらず私自身があまり活動できなかった。
プロジェクト(特に学生)に影響が無いように前向きに仕事を行う。

報告日　2006年11月9日

週 間 報 告 書

プロジェクト名　g-mod
報告者：　谷田部　学

報告期間	2006年11月2日 ～ 2006年11月8日			
1. 先週のからの継続	No.	内容	担当	状態(完了・継続・未着手)
	1	プロジェクト提案書作成	佐藤	継続
	2	詳細設計書作成	平澤	継続
	3	中間報告に向けての資料準備	谷田部	完了
	4	Q&A対応	平澤	完了
	5			
2. 実績	No.	内容	担当	状態(完了・継続・未着手)
	1	週間報告書作成	谷田部	完了
	2	進捗報告資料作成	谷田部	完了
	3	テスト仕様書作成	松田	継続
	4			
	5			
3. 課題・対策	1. プロジェクト提案書の修正			
	鳴津先生とお会いすることが出来なかったが、メールにてプロジェクト提案書を精査して頂いた。			
	修正点がいくつかあったので随時修正する。			
	2. クライアントとの連絡スケジュール			
	技術に関する質問も含めて遠峰さんと打ち合わせを行う。			
	鳴津先生は最低月に1回お会いできるようにスケジュール調整して頂くように提案する。			
4. 今週予定	1. プロジェクト提案書 修正と承認			
	2. 詳細設計			
	3. テスト仕様書作成			

作業時間サマリ

No.	ジョブ名	時間(h) 谷田部	佐藤	平澤	松田	合計
1	資料作成	9.0	0.0	7.0	0.0	16.0
2	ML対応	0.5	1.0	2.0	1.0	4.5
3	フィールドワーク	0.0	2.0	0.0	2.0	4.0
4	授業	8.0	0.0	3.0	0.0	11.0
5	気にした時間	2.0	4.0	0.0	1.0	7.0
6	その他(時間外ミーティング)	0.0	0.0	5.0	0.0	5.0
	合計	19.5	7.0	17.0	4.0	47.5

所感

チーム全体が中間報告会の為の資料作成に時間を費やしていた為にプロジェクトの進捗が無かった。

報告日　2006年11月16日

週 間 報 告 書

プロジェクト名　g-mod
報告者：　谷田部　学

報告期間	2006年11月2日 ～ 2006年11月15日			
1. 先週のからの継続	No.	内容	担当	状態(完了・継続・未着手)
	1	プロジェクト提案書作成	佐藤	継続
	2	詳細設計書作成	平澤	継続
	3	テスト仕様書作成	松田	継続
	4			
	5			
2. 実績	No.	内容	担当	状態(完了・継続・未着手)
	1	週間報告書作成	谷田部	完了
	2	進捗報告資料作成	谷田部	完了
	3	プロジェクトに関する質問5つ	全員	継続
	4	プロジェクト提案事項(案)を考える	全員	継続
	5	実現性テスト2(TCPソケット通信PG、	佐藤	継続
	6	DB2 インストールからテーブル作成まで)	全員	継続
	7	再スケジュール	谷田部	完了
3. 課題・対策	1. プロジェクト活動			
	知らぬ間にプロジェクトが下請になっていた。			
	中間報告を通して気付いたプロジェクトの方向を修正して、見直しと提案を行う。			
	2. クライアントとの連絡スケジュール			
	遠峰さんとは週に2回、時間を取ることが出来るとのこと。松田くんが遠峰さんの講義を受講していることもあり、			
	比較的接しやすい。(毎週火、木　対応可能)			
	鳴津先生は2週間に1回は授業に参加されるので、遠峰さんと打ち合わせた内容をサマリしてご確認頂く。			
4. 今週予定	1. プロジェクト提案書 修正と承認			
	2. 再スケジュール			
	3. プログラム設計書作成			
	4. テスト仕様書作成			

作業時間サマリ

No.	ジョブ名	時間(h) 谷田部	佐藤	平澤	松田	合計
1	資料作成	2.0	0.0	3.0	0.0	5.0
2	ML対応	1.0	1.0	1.0	1.0	4.0
3	フィールドワーク	0.0	0.0	0.0	0.0	0.0
4	授業	5.0	5.0	5.0	5.0	20.0
5	気にした時間	0.0	3.0	1.0	0.0	4.0
6	その他(時間外ミーティング)	1.5	0.0	0.0	1.5	3.0
	合計	9.5	9.0	10.0	7.5	36.0

所感

中間報告後、チームミーティングを行い、プロジェクト活動の誤りに全員が気付いた。
まずは今までの間違いがないようにステークホルダを意識したプロジェクト活動が出来ることを目指して、最高のプロジェクトに変身すべく案をだしあった。
今回は軽率なメールのやり取りで戸並さんには大変ご迷惑をお掛けし、大変申し訳なく感じています。
最終報告では堂々とした立派なプロジェクトになった事を証明できるようなプロジェクトになるように頑張ります。

報告日　2006年11月30日

週 間 報 告 書

プロジェクト名　g-mod
報告者：　谷田部　学

報告期間	2006年11月16日 ～ 2006年11月29日			
1. 先週のからの継続	No.	内容	担当	状態(完了・継続・未着手)
	1	プロジェクト提案書作成	佐藤	継続(停止状態)
	2	詳細設計書作成	平澤	継続
	3	テスト仕様書作成	松田	継続
	4	プロジェクトに関する質問5つ	全員	完了
	5	プロジェクト提案事項(案)を考える	全員	完了
	6	実現性テスト2(TCPソケット通信PG、	佐藤	完了
	7	DB2 インストールからテーブル作成まで)	全員	完了
2. 実績	No.	内容	担当	状態(完了・継続・未着手)
	1	週間報告書作成	谷田部	完了
	2	進捗報告資料作成	谷田部	完了
	3	11月20日　遠峰さんとのヒアリング	全員	完了
	4	11月22日　鳴津先生とのヒアリング	全員	完了
	5	11月28日　g-modミーティング	全員	完了
	6	11月29日　遠峰さんとのヒアリング	全員	完了
	7			
	8			
3. 課題・対策	1. プロジェクト残タスク			
	重要な残タスクが残っており、フェーズ毎の達成感が得られないままズルズルと来てしまっています。			
	12月21日にはシステムを納品する予定なのでそれに向けて着実に進んで行くように、スケジュール管理の徹底と			
	課題点を明確にしていきます。(期限:12月7日)			
	2. PMがクライアントと話せていない			
	学生達は非常に積極的に活動してくれたのだが、全ての打ち合わせに私が参加できていない。			
	どうしても漏れのあるヒアリングとなっているので、不明確な点を明確にして潰していくよう指示する。(期限:12月7日)			
	3. 仕様変更			
	仕様変更が入った。 OSがWindowsからUnixへ変更。 DMC側が作成するモジュールから渡される値が変更された。			
	仕様変更 が入った時に学生にコミットしないように注意する。(期限:11月30日)			
	4. スケジュール			
	提案分のスケジュールが立てられていない。学生と話し合った上で提案分を1月に対応するか判断する。			
4. 今週予定	1. プロジェクト提案書(要件確認 書)			
	2. プログラム設計書作成			
	3. テスト仕様書作成			
	4. 鳴津先生からのヒアリング(11月30日)			

作業時間サマリ

No.	ジョブ名	時間(h) 谷田部	佐藤	平澤	松田	合計
1	資料作成	4.0	0.0	6.0	1.0	11.0
2	ML対応	0.5	3.0	5.0	2.0	10.5
3	フィールドワーク	0.0	6.0	4.0	4.0	14.0
4	授業	8.0	3.0	3.0	3.0	17.0
5	気にした時間	0.0	6.0	1.0	2.0	9.0
6	その他(時間外ミーティング)	0.0	0.0	0.0	0.0	0.0
	合計	12.5	18.0	19.0	12.0	61.5

所感

自社の仕事が多忙となり、クライアントである鳴津先生や遠峰さんと会って話をすることが出来ていない。
12月も引き続き多忙を極める状態です。 g-modに携う事が出来ず学生さんには申し訳無く思います。
学生さんが積極的に活動してくれているおかげでプロジェクトを徐々に進めることができてとても感謝しています。

報告日　2006年12月7日

週 間 報 告 書

プロジェクト名　g-mod
報告者：　谷田部　学

報告期間	2006年11月30日 ～ 2006年12月7日					
1. 先週のからの継続	No.	内容	担当	時間	ジョブID	状態(完了・継続・未着手)
	1	プロジェクト提案書作成	佐藤	1	1	継続
	2	詳細設計書作成	平澤	12.5	1	継続
	3	テスト仕様書作成	松田	2	1	継続
	4	プロジェクト計画 書作成	谷田部	5	1	継続
	5	技術調査　TCPソケット通信	佐藤	3	4	完了
2. 実績	No.	内容	担当	時間	ジョブID	状態(完了・継続・未着手)
	1	週間報告書作成	谷田部	0.5	1	完了
	2	進捗報告資料作成	佐藤	1	1	完了
	3	11月30日　鳴津先生とのヒアリング	全員	1	4	完了
	4	12月7日　遠峰さんとのヒアリング	全員	1.5	6	完了
	5	12月7日　g-modミーティング	全員	1	6	完了
	6					
3. 課題・対策	1. プロジェクト遅延と学生のモチベーション低下の不安					
	プロジェクトが大幅な遅延となり学生のモチベーションがさがるのではないかと不安だったが、					
	今までより1も前向きに取り組んでいる。					
	2. ヒアリングの目的が不明確					
	先週の進捗報告や進捗報告後に行った鳴津先生とのヒアリング時に目的が不明確だと指摘された。					
	今後、ヒアリング時の目的と達成を明確にするように心がける。					
4. 今週予定	1. プロジェクト提案書(要件確認 書)　承認					
	2. プログラム設計書　承認					
	3. テスト仕様書作成					
	4. コーディング					

作業時間サマリ

No.	ジョブ名	時間(h) 谷田部	佐藤	平澤	松田	合計
1	資料作成	5.0	2.0	12.5	5.0	24.5
2	ML対応	0.5	1.0	2.0	1.0	4.5
3	フィールドワーク	0.0	0.0	0.0	0.0	0.0
4	授業	8.0	4.0	4.0	4.0	20.0
5	気にした時間	0.0	3.0	10.5	2.5	16.0
6	その他(時間外ミーティング)	0.0	3.5	3.5	3.5	10.5
	合計	13.5	13.5	32.5	16.0	75.5

所感

報告日　2006年12月14日

週 間 報 告 書

プロジェクト名　g-mod
報告者：　谷田部　学

報告期間	2006年12月7日 ～ 2006年12月13日					
1. 先週のからの継	No.	内容	担当	時間	ジョブNo.	状態(完了・継続・未着手)
	1	プロジェクト提案書提示	佐藤		1	継続
	2	プログラム仕様書への合意	平澤		1	完了
	3	テスト仕様書作成	松田		1	継続
	4	プロジェクト計画書作成	谷田部		1	継続
	5					
2. 実績	No.	内容	担当	時間	ジョブID	状態(完了・継続・未着手)
	1	週間報告書作成	谷田部		1	完了
	2	進捗報告資料作成	平澤		1	完了
	3	12月7日　g-modミーティング	全員		4	完了
	4					
	5					
	6					
3. 課題・対策	1. 開発遅延と冬期休暇					
	開発が遅れているが、何とか年内にはクライアントテストを実施できる体制を取りたいとの事でした。					
	あと10日間学生のみんなには頑張ってもらいたい。					
4. 今週予定	1. プロジェクト提案書(要件確認書)　再提出					
	2. テスト仕様書作成					
	3. コーディング					

作業時間サマリ

No.	ジョブ名	時間(h) 谷田部	佐藤	平澤	松田	合計
1	資料作成	2.0	4.0	13.5	9.0	28.5
2	ML対応	0.5	3.0	2.0	1.0	6.5
3	フィールドワーク	0.0	1.0	1.0	0.0	2.0
4	授業	8.0	3.0	3.0	3.0	17.0
5	気にした時間	2.0	2.5	2.0	2.0	8.5
6	その他(時間外ミーティング)	4.0	2.0	2.0	2.0	10.0
	合計	16.5	15.5	23.5	17.0	72.5

所感

前回の授業後、g-modで飲み屋に行ってきました。
プロジェクト進捗遅延からメンバー一同危機感を感じ続けている状態だったので、モチベーションを心配していました。
私が思っていた以上に元気で前向き。まだまだ自分達の力でヤリキル気持ちが感じられたのが何よりの収穫です。
プレッシャー・危機感を常に感じており、精神的に辛い日々が続いていますが色々なことが起こる楽しいプロジェクトです。

報告日　2006年12月21日

週 間 報 告 書

プロジェクト名　g-mod
報告者：　谷田部　学

報告期間	2006年12月14日 ～ 2006年12月20日					
1. 先週のからの継	No.	内容	担当	時間	ジョブNo.	状態(完了・継続・未着手)
	1	プロジェクト提案書　再提出	佐藤		1	継続
	2	プログラム仕様書への合意	平澤		1	完了
	3	テスト仕様書作成	松田		1	継続
	4	プロジェクト計画書作成	谷田部		1	継続
	5					
2. 実績	No.	内容	担当	時間	ジョブID	状態(完了・継続・未着手)
	1	週間報告書作成	谷田部		1	完了
	2	進捗報告資料作成	松田		1	完了
	3	12月18日　遠峰さんとの打合せ	全員		4	完了
	4	コーディング	全員		1	継続
	5					
	6					
3. 課題・対策	1. 冬期休暇に向けて					
	メンバーの冬期休暇予定は下記の通り。					
	佐藤:12月25日～　不在(国内)　携帯電話で連絡可能。					
	平澤:12月23日～1月6日　不在(海外)　Webmailで連絡可能。					
	松田:自宅　携帯電話、メールで連絡可能。					
	冬期休暇中は松田くんがメインで残作業を行う。途中、佐藤くんが神奈川に戻ってきて松田くんのフォローをする可能性あり。					
4. 今週予定	1. コーディング					
	2. レビュー(松田さん)					
	3. 単体テスト、結合テスト					
	4. リリース環境構築					

作業時間サマリ

No.	ジョブ名	時間(h) 谷田部	佐藤	平澤	松田	合計
1	資料作成	4.0	24.0	12.0	12.0	52.0
2	ML対応	0.1	0.5	0.5	0.5	1.6
3	フィールドワーク	0.0	0.0	0.0	0.0	0.0
4	授業	8.0	3.0	3.0	3.0	17.0
5	気にした時間	0.0	0.0	0.0	0.0	0.0
6	その他(時間外ミーティング)		2.0	1.0	2.0	5.0
	合計	12.1	29.5	16.5	17.5	75.6

所感

コーディングに入ってプロジェクトは学生達に任せている。
3人のチームワークも良くなってきているが、あせりも感じる。
明日の進捗次第では、リリースを遅らせる事も考える。

報告日　2007年1月11日

週 間 報 告 書

プロジェクト名　g-mod
報告者：　谷田部　学

報告期間	2006年12月21日 ～ 2007年1月10日					
1. 先週のからの継	No.	内容	担当	時間	ジョブNo.	状態(完了・継続・未着手)
	1	プロジェクト提案書　再提出	佐藤		1	継続
	2	テスト仕様書作成	松田		1	継続
	3					
	4					
2. 実績	No.	内容	担当	時間	ジョブID	状態(完了・継続・未着手)
	1	週間報告書作成	谷田部	0.5	1	完了
	2	進捗報告資料作成	谷田部	1.5	1	完了
	3	12月21日　本番環境構築	全員	3	1	継続
	4	ソースマージ	全員	5	1	継続
	5	テストモジュール作成	全員		1	継続
	6	単体テスト	全員		1	継続
	7	結合テスト	全員		1	未着手
3. 課題・対策	1. 冬期休暇中の遅延					
	年末から年始に掛けてバッタリと活動が停止してしまった。					
	停止していた時間を取り戻す為にスケジュールを再作成した。					
4. 今週予定	1. レビュー					
	2. 単体テスト、結合テスト					
	3. リリース環境構築					
	4. 操作マニュアル作成					

作業時間サマリ

No.	ジョブ名	時間(h) 谷田部	佐藤	平澤	松田	合計
1	資料作成	2.0	13.0	12.5	12.0	39.5
2	ML対応	0.1	0.5	0.5	0.5	1.6
3	フィールドワーク	0.0	2.0	2.0	2.0	6.0
4	授業	8.0	3.0	3.0	3.0	17.0
5	気にした時間	2.0	1.0	2.0	1.0	6.0
6	その他(時間外ミーティング)	0.5	0.0	0.0	0.0	0.5
	合計	12.6	19.5	20.0	18.5	70.6

所感

年末は佐藤くんからのテスト報告を最後に作業が止まってしまった。
12月は特にプレッシャーが大きかったことや作業時間が多かったことにより、精神的にもお休み状態になってしまったと感じている。
早々に作業を復帰し納品に向けて追い込みを掛ける。

人材 募集！

ネクストウェア株式会社
谷田部 学

募集人材

- 必要スキル　特になし
- 責任感とやる気があれば誰でもOK
- とにかく話しましょう！

こらまね　テーマ

- DMC　GeocodingModule開発
- クライアント：　嶋津　恵子先生

スケジュール案

年	2006												2007							
月	10				11				12				1				2			
木曜日	1	2	3	4	1	2	3	4	5	1	2	3	4	1	2	3	4	1	2	3	4
全体						▲中間発表		▲三田祭										▲最終発表会（予定）			
プロジェクト		▲仕様概要		▲プログラム仕様の合意					▲テスト仕様の合			▲納品									
要件定義																					
外部設計																					
詳細設計																					
テスト設計																					
開発																					
単体テスト																					
結合テスト																					
ユーザ検証																					
納品												▲									
備考	週の右側の線を開始とする 例えば、要件定義は1週目木曜日～3週目木曜日となる																				

プロジェクト指針（仮）

- PJテーマ

  使用されるシステムを開発しよう

  ↓その為には

  ↓

品質管理の強化

  品質基準、コーディング規約、テスト手法

サマリ

先週の実績

1. チーム決定
2. 要件のヒアリング
3. WBS作成
4. スケジュール作成
5. 課題

会社概要と自己紹介

- ネクストウェア株式会社
  URL：http://www.nextware.co.jp

- 技術統括本部　東京技術部
  アシスタントマネージャ　谷田部学
  入社7年
  客先常駐してのシステム開発から受託案件を行ってきた。
  言語：C、Java、VB

1. チーム決定

10月5日

N·e·x·t·W·a·r·e

進捗報告
2006年10月5日～2006年10月12日

プロジェクト：ｇ－ｍｏｄ
谷田部
佐藤
平澤
松田

チーム決定

平澤くん

佐藤くん

松田くん

よろしく！

2月28日

チーム名は？

最終発表会（予定）まで

ではなく

g-mod

ジーモード

ジーモド

10月4日

g-modロゴ

要件ヒアリング

DMCネットワークシステム Map型UI表示機能 機能仕様書

してきました。

GET！

GeocodingModule 要求仕様書

戸波さん

嶋津先生

4名で

助手
斉藤さん

10月11日

約1時間

WBS作成

要件ヒアリング

後で配ります

後で配ります

課題

スケジュール作成

要件定義書作成

要件ヒアリング

以上。

パート２

① 10/19日までに実施した活動

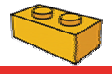
- 嶋津先生からの要件ヒアリング
  - 次頁から詳しく説明します
- Wikiページの整備
- 目標確認

g-mod班進捗報告

2006/10/19

PM 谷田部学

佐藤俊之　平澤秀幸　松田航



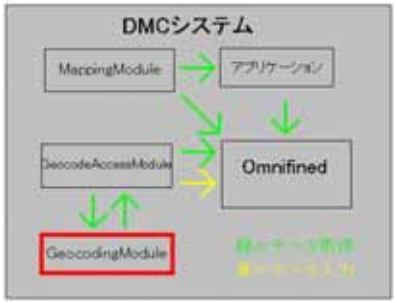
要件ヒアリングの目的・日時

- 目的（聞きたかった内容）
  - 資料について
  - 仕様について
  - やりとりの方法について
  - 感じた疑問・質問
- 日時
  - 10月12日の4限の間
- 参加者
  - g-modチーム、嶋津先生、遠峰さん（村井研の方）

目次

要件ヒアリングの結果

図にするとこんな感じ



注・資料がないので名称が
間違っていることがあります

- Omnifined
  - 索引(コレクション)を持ち、これを対象に検索する
- MappingModule
  - 地理情報に基づき、検索結果を地図を表示する
- GeocodingModule
  - 住所データリストをから、地理情報リスト(緯度経度)をつくる
- GeocodeAccessModule
  - 索引から住所データリストを取得し、GeocodingModuleに渡す
  - つくってもらった地理情報を索引に入れる

CreW
Creative Workspace

要件ヒアリングの結果

- 資料について
  - これまで頂いていた資料が間違っていた
    - 要求仕様書・機能仕様書
  - そのため、仕様が解釈していたものと変わった
    - システム構成に変更があった
      - 地理情報取得関連のモジュールが増えた
      - これにより、システムの仕様が少し変わった
  - 資料は修正中で、後日配布するとのこと
    - すぐに配布できると言われたのだが、精査中とのことでまだ貰えていない

CreW
Creative Workspace

要件ヒアリングの結果

- つまり、GeocodingModuleとは？
  - 住所データリストから、地理情報リストをつくるもの

CreW
Creative Workspace

要件ヒアリングの結果

- 新たな仕様について(おさらい込む)
  - 今回我々が作るのはDMCシステム内のGeocodingModuleというモジュール
    - DMCシステム全体は、発想支援を目的として、コンテンツ検索やコンテキスト発見のために用いるためのもの
    - 今回作成するGeocodingModuleは、地図を表示するのに必要な地理情報を取得するためのもの
      - 「地図を表示させる」モジュールではない

CreW
Creative Workspace

要件ヒアリングの結果

- 感じた疑問
  - 先述したもの以外にも疑問点があった
  - このモジュールの目的は『DMCシステムの目的である「知の再編」の支援』とあるが、「知の再編とはどういう意味？」
    - 見つかった知識体系をいままでにない知識体系に結びつかせ、発想の支援とする

要件ヒアリングの結果

- 細かい仕様
  - 住所データから地理情報（緯度経度）を割り出す
    - 公開されているWeb上のシステムを使用する
  - 一度取得したものはキャッシュに入れておく
    - 2度目以降は取得の必要を無くす
  - 一定の期間が過ぎたらキャッシュをクリアする
    - 負荷が未知数なので、期間の設定ができるように
  - 地理情報データをGeocodeAccessModuleに返す

ヒアリングの感想

- 感想
  - 顔合わせをすることができてよかった
  - 何をやらねばならないのかがよく見えてきた
  - 内容のあるヒアリングができたと思う

要件ヒアリングの結果

- やりとりの方法について
  - 書類の承認方法と時期
  - 技術的な質問が生じた場合の連絡先

g-mod班進捗報告

2006/10/26
PM 谷田部学
佐藤俊之　平澤秀幸　松田航

②今後の予定

- 詳細設計
  - 要件定義書の作成
    - 目次は作成済み、今日この後検討する予定
  - 今後のスケジュールの検討・確認
    - 嶋津先生との都合もあり、検討のし直しが必要
    - その後、改めて来週までのタスクを洗い出す

CreW
Creative Workspace

目次

- 今までの流れ
- 今週の活動内容
- 今後の活動内容
- 作業時間
- 一週間の感想

以上です

ご清聴ありがとうございました

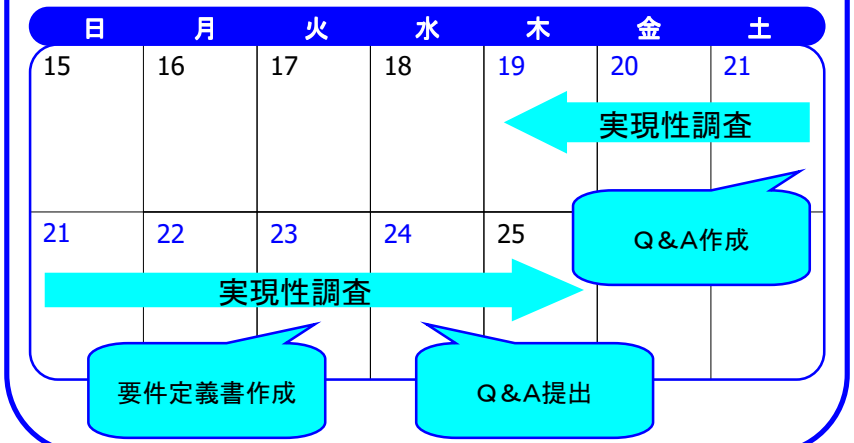
CreW
Creative Workspace

今までの流れ

- 要件ヒアリングの結果
  - 頂いた資料が間違っていた
    - 修正版をヒアリングの翌日に頂く予定だった
  - DMCシステムのGeocoding Moduleを作って欲しい
  - 感じたこと
    - 顔合わせをすることが出来てよかった
    - 内容のなるヒアリングが出来た

今までの流れ

今週の活動内容



今までの流れ

- 要件ヒアリング
  - 日付： 10月12日
  - 参加者
    - g-modチーム・嶋津先生・遠峯さん
  - 目的
    - 配布資料と仕様について
    - やり取りの方法
    - 感じた疑問質問

Q&A作成

- 目的
  - 現状のDMCシステムの構造を把握する
  - 要件の抽出及び再確認
  - 配布資料の内容確認
- 質問内容
  - 用語定義（配布資料で使われていた言葉の意味）
  - システム構造（現在のシステム構造に関する質問）
  - その他
    - 例） 我々のプロジェクトの他に期待する事はありませんか。

グループミーティング

- 日付： 2006/10/19
- 内容
  - PMから修正版の要件定義書とシステム設計書を入手
  - スケジュールの確認
  - その他
    - 作業時間の報告方法

要件定義書作成

- 配布資料を基に作成
- 目次
  - Ⅰ. システムの目的
  - Ⅱ. 機能仕様書
  - Ⅲ. 要求定義書
  - Ⅳ. 作業範囲
  - Ⅴ. 開発環境
  - Ⅵ. 品質基準テスト
  - Ⅶ. スケジュール
  - Ⅷ. 納品物

グループミーティング

- 決まったこと
  - 品質重視にプロジェクトを進める

作業時間

- PM　谷田部　学
- 学生メンバー
  - 佐藤:
  - 平澤:
  - 松田:

実現性調査

- 目的
  - システムの実現性を検証
- 項目
  - Geocoding APIの使用法
  - JavaにおけるXMLの扱い方
  - DBへのアクセス方法
  - 外部定義ファイルの読み込み

一週間の感想

- メンバー間の認識のずれがあった
  - お互いに情報交換をすることにより直すことが出来た

今後の活動内容

- 要件定義:10月 5日～10月25日
  - 目標:　11月2日承認
- 詳細設計:10月19日～10月27日
  - 7日遅延
- 実現性調査:10月19日～10月27日
  - オンスケジュール
- テスト設計:10月27日～11月15日

g-mod

目次

I. 以前のg-mod

II. 気づき

III. どうする

IV. スケジュールの変更

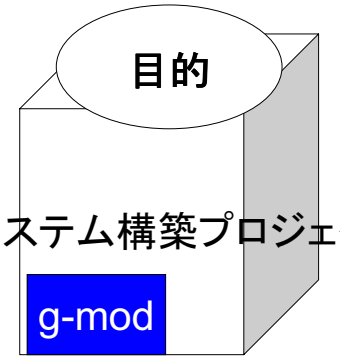
V. 終わりに

以上です

ご清聴ありがとうございました

g-mod

Ⅰ．以前のg-mod



g-mod 進捗報告

2006年11月16日

谷田部学
佐藤俊之
平澤秀幸
松田航

KEIO University DMC
CreW Creative Workspace

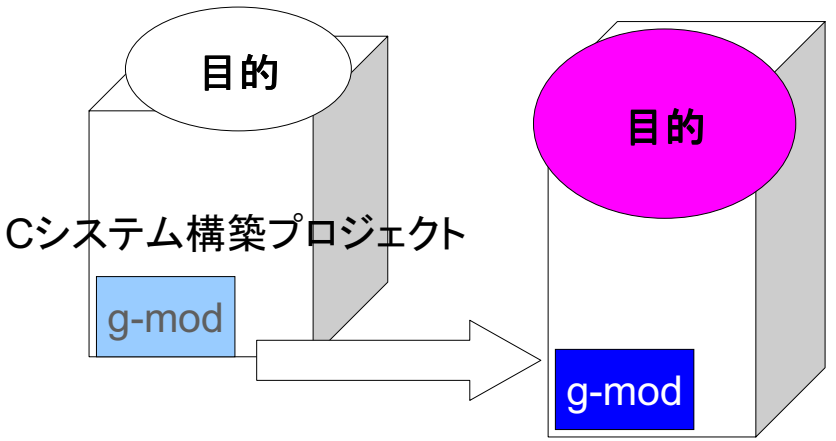
g-mod

Ⅰ. 今までのg-mod

- とにかくクライアントの指定通りのモノを作る。
  - 実運用されるモノを作って納品しよう。
  - クライアントを納得させるモノを作ろう。
- 何をするのか理解されていない不安。
  - なぜ理解されないのか。
  - とにかくわからない事を解消したい

g-mod

Ⅱ. 気づき



g-mod

Ⅱ. 気づき

- とにかくクライアントの指定通りのモノを作る。
  - 他のステークホルダーが意識されていない。
  - 下請になっていた。
- 何をするのか理解されていない不安。
  - 何も提案していない。

g-mod

Ⅲ. どうする

- プロジェクトの見直し。
  - ステークホルダーに夢を見せる。
  - もう一度プロジェクトの内容を見直す。
  - どのような提案をするか。
  - クライアントから話を引き出す。
- 現状のプロジェクトを進める。
  - 現状の不安要素を解決する。
- 再スケジュール

11月16日～11月29日
進捗報告

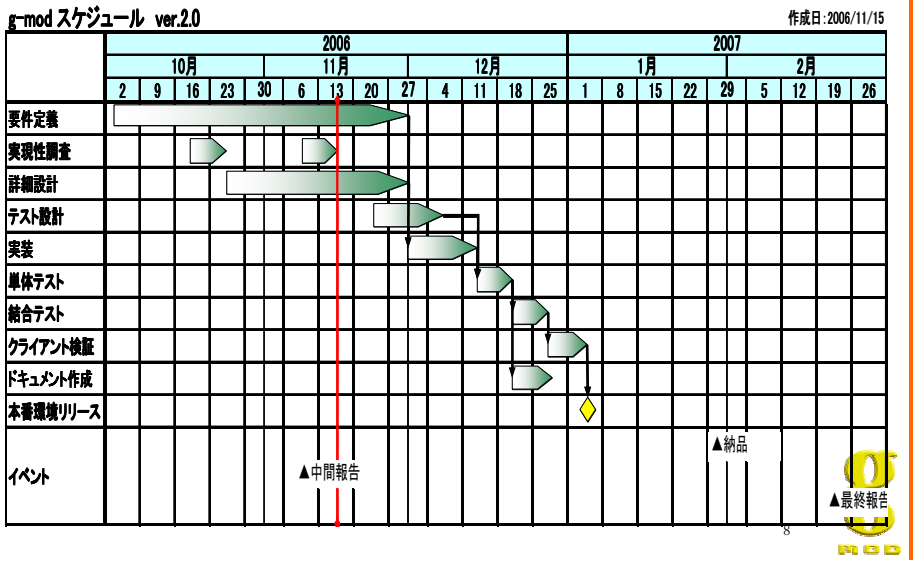
2006年11月30日

谷田部学
佐藤俊之
平澤秀幸
松田航

KEIO University DMC
CreW
Creative Workspace

g-mod 目次

g-mod Ⅳ. スケジュール変更



g-mod Ⅴ. 終わりに

最終報告会ではg-modが
変わった事を　戸並さん　及び
出席者の皆さんに見せるぞ！！

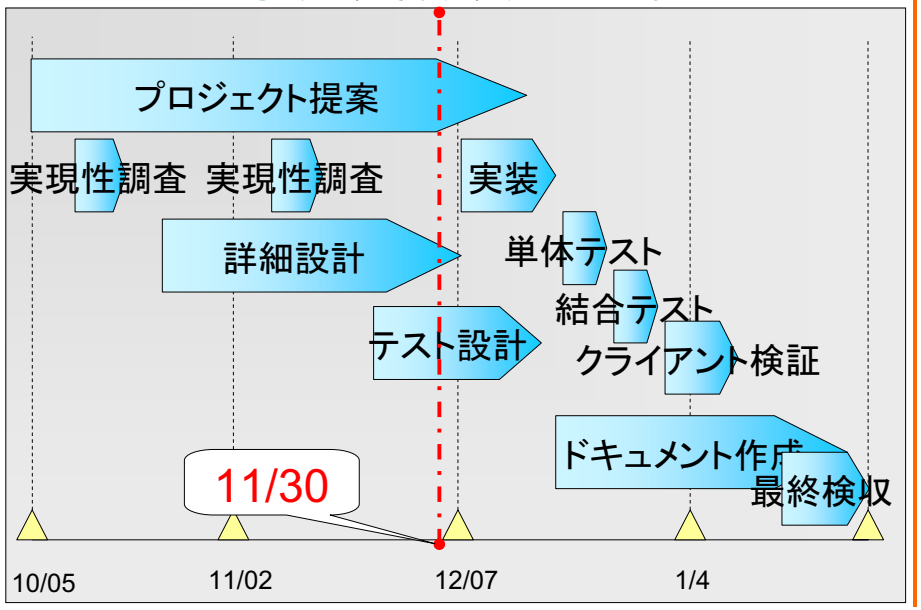
g-mod

実績の概要

- 11月20日　遠峰さんミーティング
  （DMC）遠峰さん　（g-mod）佐藤、松田
- 11月22日　嶋津先生ミーティング
  (DMC)嶋津先生　(g-mod)佐藤、平澤
- 11月28日　g-mod内部ミーティング
  (g-mod)佐藤、平澤、松田
- 11月29日　遠峰さんミーティング
  (DMC)遠峰さん（g-mod）佐藤、平澤、松田
- プログラム仕様書作成
- テスト仕様書作成

g-mod

ざっくりスケジュール



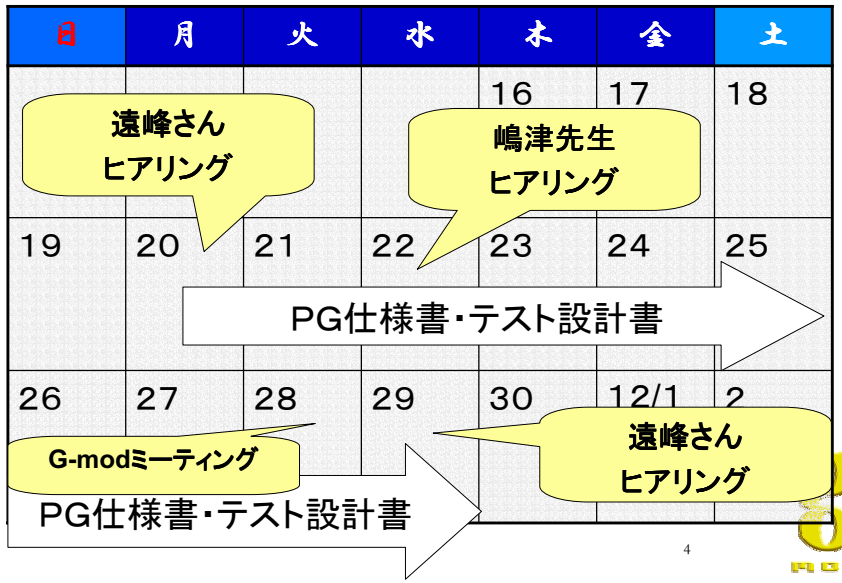
g-mod

11月20日
遠峰さんミーティング

1. 要求事項についての確認と提案
   - 緯度経度算出モジュールではなく
     緯度経度算出システムを作って欲しい！
2. 詳細仕様（機能）についての質問
   - データのやり取り
     - ソケット通信なところは変わらない。
   - エラーログ
     - どんな形でもいいので、確認できるようログに残す
   - 返す値はnullとか？
     - エラーコードを決める必要がある（ログ及び返す値用）
     - DMC側で指定する、という形になった（はず･･･）
   - timeout
     - 「接続できなかった」と判断する時間は何秒くらいか
       ⇒プロパティファイルで設定できればいいのでは。
     - 5秒くらいでいいと思う、という案には賛成。
   - DMCシステム（GeocodingAnotator）自体はいつできるか
     - g-modのが出来上がるまでにはできる。がんばる。

g-mod

ざっくり実績



g-mod

11月29日
遠峰さんミーティング

1. 詳細仕様について

・緯度経度算出システム

DMCシステムの一部として稼動するわけではなく全く別のマシンにインストールする。

・OS

遠峰さんとしてはUNIXマシンを想定。

・ログ

DBではなくファイルに保存して欲しい。

デバッグログと通常ログの切り分け

・DB初期データ

現在DMCシステムが緯度経度算出に使用しているCVSファイルのデータを移行する。

・複数の地名があるときは工夫する

g-mod

11月22日
嶋津先生ミーティング

1. 要求事項についての確認と提案
   - 新しく何かを提案してくれるのは大歓迎
   - 新しく提案するのなら、それが実現できる証拠(evidence)を提示して欲しい
   - この時期で終わるのかどうかが心配
2. 詳細仕様についての質問
   - DMCの現状として、現在マップに検索結果を表示させる事は出来るが、マップ表示に必要な緯度経度情報は手打ちでCSVファイルに書き込んでいる

→数が多いと手間がすごく掛かる

→人だから間違える可能性もある

→ここをなんとか自動化したい

   - Map型UIの構造について

・システムの将来性を考え、独立性と再利用性がある設計にした

・Omnifindに組み込まない設計にした

g-mod

まとめ

- クライアント
  - まずは要求に合ったものを作る
  - 提案は大歓迎
  - この期間内に終わるのか不安
- システム開発
  - 緯度経度算出モジュールから緯度経度算出システムへ
  - 機能詳細の把握と不明点の明確化
- 目的と目標
  - チーム内で目的、目標の認識を統一

g-mod

11月28日
g-mod内部ミーティング

1. 目的と目標

【目的】 慶應義塾大学デジタルメディアコンテンツ統合システムで**使ってもらえる新しいシステムを開発する**

【目標】 品質の高いシステムを構築する

2. プロジェクト学習目標

g-modとして何を学ぶのか？

3. 11月29日遠峰さんとのヒアリング準備

・どういう形で緯度経度情報を返して欲しいか？

・エラー処理のコード規約

・キャッシュテーブルのレイアウト

・DBは誰がDMCのサーバにインストールか？

・動作概要をレビューしてもらう

g-mod

PMの心配事（その他）

ミス慶應 ☆ 竹内由恵

ミスター慶應 ？　誰だったか知ってる人？

g-mod

今後の動き

- 詳細設計（プログラム仕様書作成）
  - １１月２９日ヒアリング結果からプログラム仕様書に修正を加えて嶋津先生へ提出したい。
- テスト設計（テスト仕様書作成）
  - 単体テストまでにテスト項目を洗い出す
  - 作成するテストモジュール洗い出し
- 開発環境構築
  - postgreSQL
  - J2SDK1.5

12月07日～12月13日　進捗報告

2006年12月14日

谷田部学
佐藤俊之
平澤秀幸
松田航

KEIO University DMC
CreW Creative Workspace

g-mod

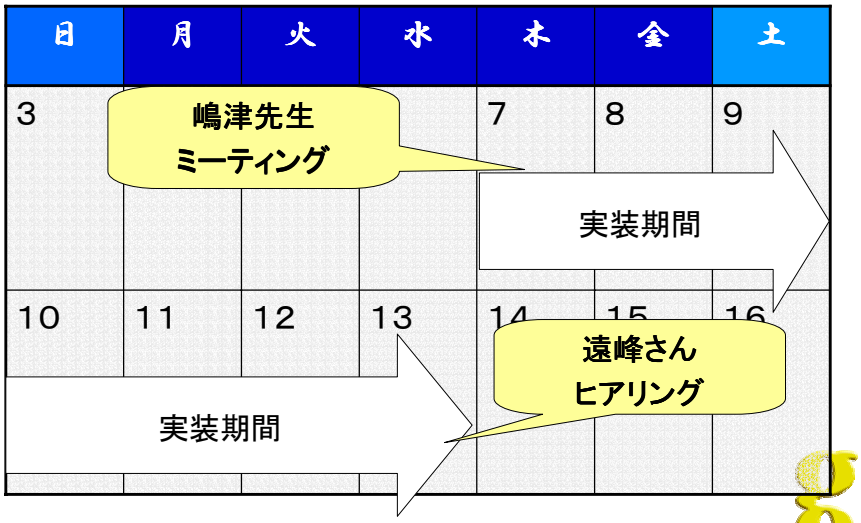
作業時間11月16日～29日

単位：時間

	佐藤	平澤	松田	谷田部
資料作成	1	6	2	5
ML対応	4	5	1	5.5
フィールドワーク	4	4	4	0
授業	3	3	3	8
気にした時間	2	1	2	1

g-mod

2006/12/07～2006/12/13



g-mod

目次

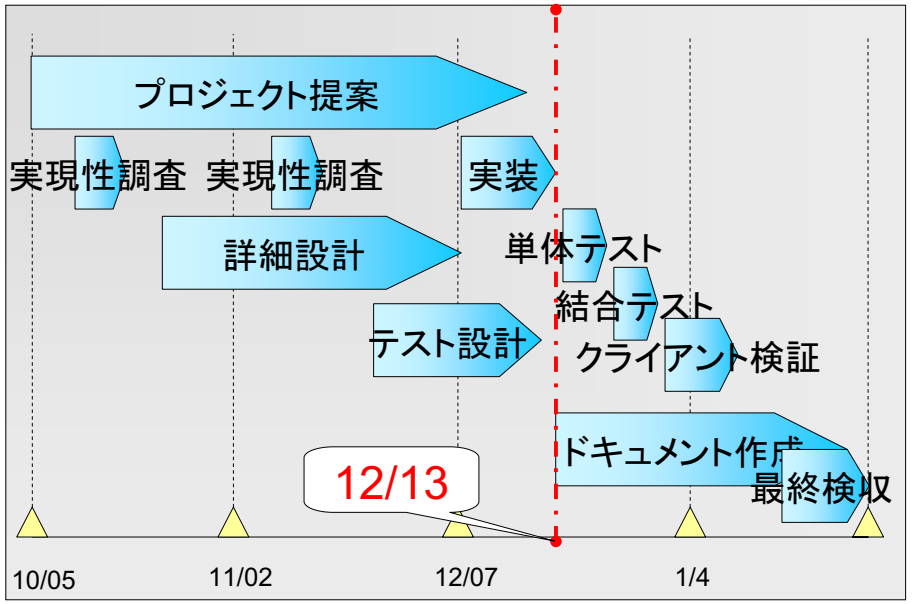
g-mod

実績（時系列順）

- 12月07日　嶋津先生ミーティング
  - DMC：　嶋津先生
  - g-mod：　全員
  - コラマネ：　松澤さん
- 12月07日　チームミーティング
  - g-mod：　全員
- 12月13日　遠峰さんヒアリング
  - DMC：　遠峰さん
  - g-mod：　佐藤、平澤

g-mod

スケジュール



g-mod

12月07日　チームミーティング

- 場所
  - 湘南台の坐和民
- 目的
  - プロジェクトの現状を分析し、問題点を洗い出す
  - 各問題の解決策を練る

g-mod

12月07日　嶋津先生ミーティング

- 方針
  - オープンディスカッション
    - 思ったことを自由に発言した
- 内容
  - 前日のヒアリングの感想
  - お互いの意識の確認
  - 合意が取れているものを確認

g-mod

12月07日　チームミーティング

- 事前準備
  - 各メンバーでプロジェクトに関して疑問に思っている点を考え、書き留めた
- 方針
  - 疑問に思っていたことを提示して、それについて話し合った
  - 解決できるものはその場で解決する

g-mod

12月07日　嶋津先生ミーティング

- 発見
  - 抽象的な表現を多く使っていた
    - 具体性がなかった
      例）「設計書」という言葉を使っていたが、何の設計書か相手に伝わらなかった
  - クライアントの言葉を素直に受け入れていた
    - 一方的にクライアントの意見を聞いていた
      ⇒「交渉」が起こらない
    - クライアントとの「キャッチボール」がなかった
      ⇒どちらかというと「ドッジボール」

g-mod

合意リスト

g-mod

12月07日　チームミーティング

- 結果
  - メンバー同士が何を考えている／心配しているかを確認できた
- 感想
  - 「お互いが考えている事を垣間見ることが できてとても楽しかったし、良かったと思います。」 谷田部

g-mod

12月13日　遠峰さんヒアリング

- 目標
  - プロジェクト提案書への合意
  - プログラム仕様書の説明
    - 入出力の仕様の合意を取る
  - プロジェクトの実現性の証明
- 方針
  - 修正したドキュメントを提示して、コメントを頂く

g-mod

プロジェクトの問題点

- ヒアリングの行い方
  - こちらからあまり問いかけなかった
    - クライアントに押されたまま、ヒアリングが終了
    - 持って帰る成果があまりなかった

    ⇒疑問に思った箇所をすぐに質問することに
  - 目的が不明確
    - クライアントもヒアリング目的が分からなかった

    ⇒事前にヒアリングレジュメを用意することに

g-mod

12月13日　遠峰さんヒアリング

- 目標
  - プロジェクトの実現性の証明
- 結果
  - 証明できた
    - 実現性調査の成果物を提示
    - 安心していただけたかと信じている

g-mod

12月13日　遠峰さんヒアリング

- 目標
  - プロジェクト提案書への合意
- 結果
  - 合意は取れなかった
  - 合意をもらえない理由を聞き出した
    - 表現上のミスが大半
      ⇒具体的にどこがおかしいか指定していただいた
    - 抜けている内容

g-mod

実装

- 最初に立てたスケジュールでは実装段階に入っている
- 各メンバーの実装担当内容は決まった
- スケジュール調整

g-mod

12月13日　遠峰さんヒアリング

- 目標
  - プログラム仕様書の説明
- 結果
  - 合意は取れた
    - 入出力の形について
  - データベースのテーブル設計に関して高評価
    - 「素晴らしい」という言葉を頂いた

g-mod

今後の活動

- 実装期間
  - 12/14~12/21
- 単体・結合テスト期間
  - 12/??~12/??

g-mod

見積もり

- プロジェクト提案書　修正（時間：）
- 遠峰さんとの打合せ（時間：）
- プログラム仕様書　修正（時間：）
- テスト仕様書　作成
- 開発環境構築
- 遠峰さんとの打合せ事前ミーティング
- 進捗報告資料作成

12月14日～12月20日　進捗報告

2006年12月21日

谷田部学
佐藤俊之
平澤秀幸
松田航

KEIO University DMC

CreW
Creative Workspace

g-mod

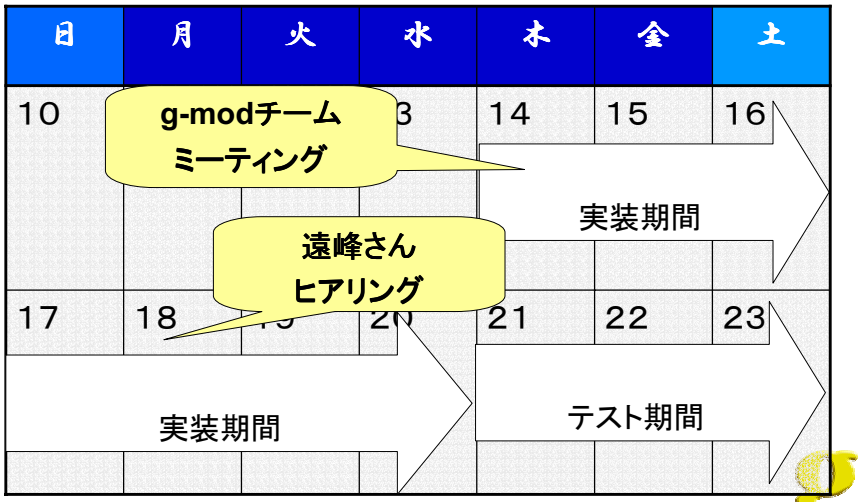
作業時間

単位：時間

	佐藤	平澤	松田	谷田部
資料作成	0	0	0	0
ML対応	0	0	0	0
フィールドワーク	0	0	0	0
授業	0	0	0	0
ミーティング	0	0	0	0
気にした時間	0	0	0	0

g-mod

2006/12/14～2006/12/20



g-mod

目次

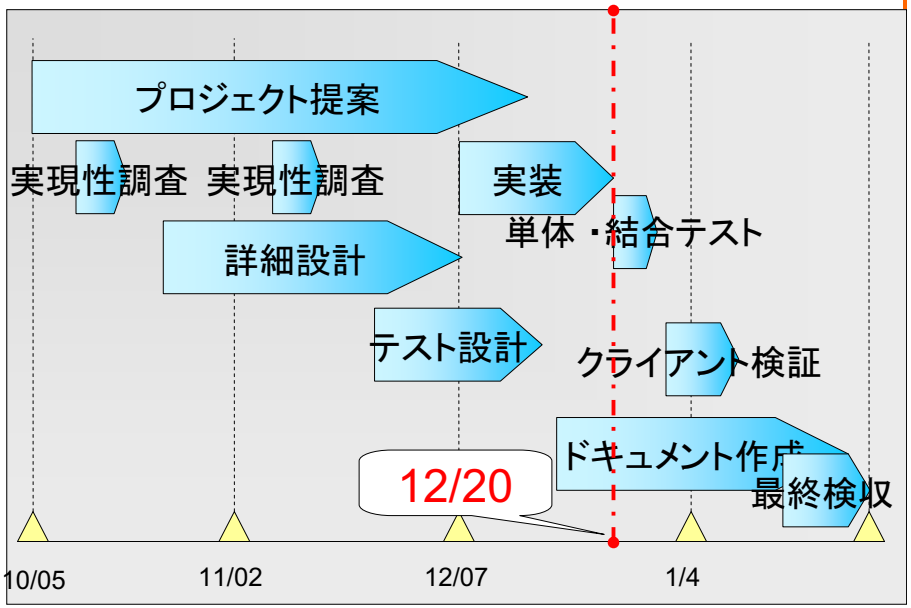
g-mod

実績（時系列順）

- 12月14日　実装開始
- チームミーティング
  - g-mod : 全員
- 12月18日　遠峰さんヒアリング
  - DMC : 遠峰さん
  - g-mod : 佐藤、松田

g-mod

スケジュール



g-mod

12月18日 遠峰さんヒアリング

- 環境構築のお願い
  - OSのインストールはDMCにお願いしたい
    - OSインストールまではDMC、それ以降はg-modがやる
  - 早い段階での構築を
    - 12月21日の4限SFCにて行う
    - 「あとは Geocoding Module を入れるだけ」という状態にする

g-mod

12月14日チーム内ミーティング

- 内容
  - 全体スケジュール確認
  - 実装スケジュール確認
  - 実装担当確認

g-mod

12月18日　遠峰さんヒアリング

- GeocodingModule の実行環境の確認
  - サーバに使用するOSは Debian GNU/Linux にすることの確認
  - ポート番号の確認
  - メモリ、ハードディスクの容量の確認
  - DBをPostgreSQLからMySQLに変更

g-mod

12月18日 遠峰さんヒアリング

- 日時
  - 12月18日 19:50～ 30分程度
- 場所
  - デルタ館北1F会議室
- 内容
  - 実装スケジュールの報告
  - 環境構築のお願い
  - GeocodingModule実行環境の確認
  - 詳細プログラム仕様の確認

g-mod

実装進捗

- 実装担当
  - 佐藤 GeocodingModule API接続、DB周り
  - 平澤 設定読み込み、本処理、通信周り
  - 松田 プログラムの動作ログを出力させる
- 見積もり時間 / 実際に掛かった時間 / 進捗率(主観)
  - 佐藤 10時間 / 24時間 / 90%
  - 平澤 15時間 / 7時間 / 100%
  - 松田 10時間 / 8時間 / 80%

g-mod

12月18日 遠峰さんミーティング

- 遠峰さんからのコメント
  - 年内に実装が難しい場合は必ず連絡する。進捗はマメに欲しい
  - ドキュメント類(インストール手順書など)が欲しい
  - TomCat をインストールするのに納得できるような理由が欲しい

g-mod

作業時間

- ヒアリング
  - 佐藤・松田：0.5時間（見積もり0.5時間）
- その他ミーティング
  - 佐藤：3時間
  - 平澤：3時間
  - 松田：3時間

g-mod

12月18日 遠峰さんヒアリング結果

- 決定事項
  - サーバのOS
  - OSインストールまではDMC側が行う（12月21日）
  - DBはMySQLを使用する
- 検討事項
  - TomCat をインストールする理由を考える（インストールすること自体を検討する）

g-mod

今週の実績

- 作業実績
  - 実装に入ることができた
  - 実装がほぼできた
- 問題点
  - ソースコードの結合作業が残っている
  - サーバのスペックを再考する必要がある

g-mod

作業時間

- PM
  - 谷田部
    - 週報0.5　　ML対応0.5
- 学生
  - 佐藤：26時間
    - 実装24　　ML対応2
  - 平澤：13時間
    - 実装7　　ML対応2　　ドキュメント修正4
  - 松田：10時間
    - 実装8　　ML対応2

g-mod

以上で発表を終わります

皆さん、良いお年を

いろいろな意味で

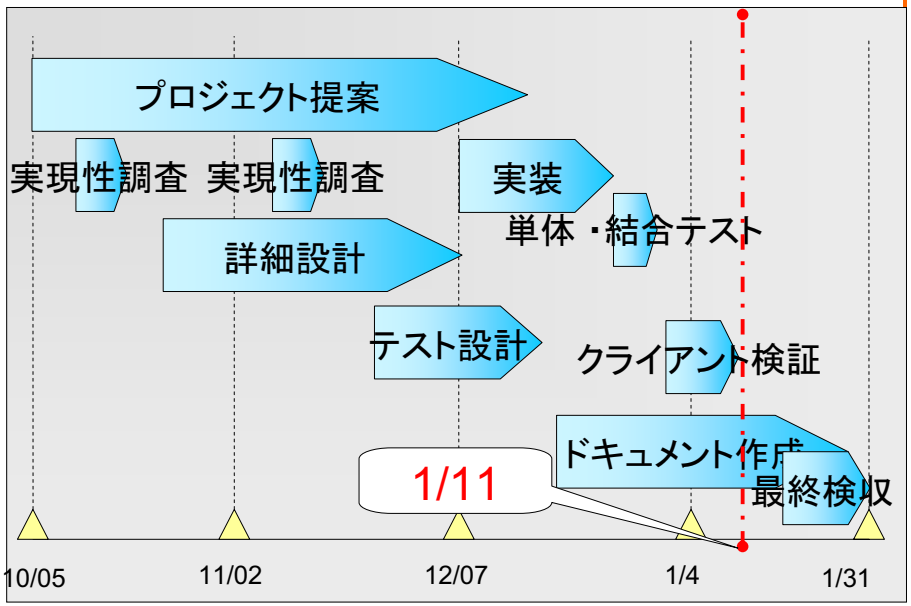
g-mod

今後の活動

- 単体、結合テスト
  - 12月21日～12月27日
- 本番環境リリース
  - 12月28日 DMC本部にて

g-mod

スケジュール



12月22日～1月10日　進捗報告

2007年1月11日

谷田部学
佐藤俊之
平澤秀幸
松田航

KEIO University DMC

CreW
Creative Workspace

g-mod

2006/12/21～2006/12/30

日	月	火	水	木	金	土
17	18	19	20	21	22	23
24	25	26	27	28	29	30

本番環境構築

テスト期間

テスト期間

g-mod

目次

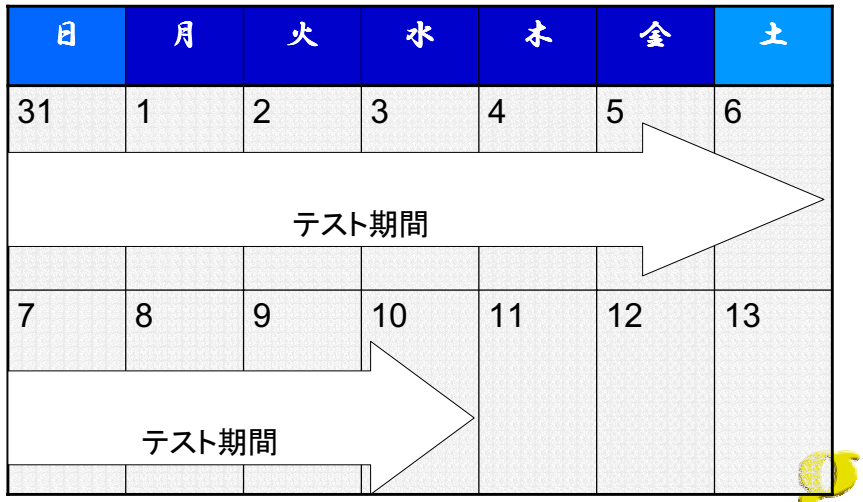
g-mod

12月21日本番環境構築

- 作業内容
  - 遠峰さんと本番環境を構築する
    - MySQLのインストール
    - Java実行環境の構築
  - 補足
    - OS（ DebianLinux ）のインストールは遠峰さんが実施した。
    - 遠峰さんのノートPCから本番環境に接続してMySQLやJava実行環境の構築を実施した。

g-mod

2006/12/31～2007/1/10



g-mod

12月21日本番環境構築

- 作業結果
  - MySQLのインストール
    →失敗：インストールしたMySQLのVersionが古い。
  - Java実行環境の構築
    →途中：環境設定が未実施となっている。

  ※時間切れで作業終了となった。
- 対応
  - 遠峰さんが本番環境を外部からアクセス出来るように設定してくれる。
  - g-modが外部からアクセスして本番環境を構築する。
- その他
  - テスト実施に際して、まだ遠峰さんとテスト仕様の合意が取れていない。
    →テスト設計書を遠峯さんに送付するので見て頂く。

g-mod

実績（時系列）

- 12月21日　本番環境構築

  時間：14:45～17:00

  場所：大学院棟（ τ 館）2F　奥
  - DMC：遠峰さん
  - g-mod ：佐藤、平澤、松田、谷田部

  目的：GeocodingModuleを配備するだけでいい環境を構築する。
- 12月22日以降
  - ソースコードの精査（マージ、コメント追加、既知の不具合修正）
  - テストモジュール作成
  - 単体テスト

g-mod

今後の活動

- 遠峰さんと連絡を取り打合せを実施
- ソースレビュー
  - 1月15日？
- 単体、結合テスト
  - 1月11日～1月14日
- テスト結果報告書作成
  - 1月15日～1月16日
- 本番環境構築
  - ？月？日
- ドキュメント作成
  - 操作マニュアル作成　～1月18日

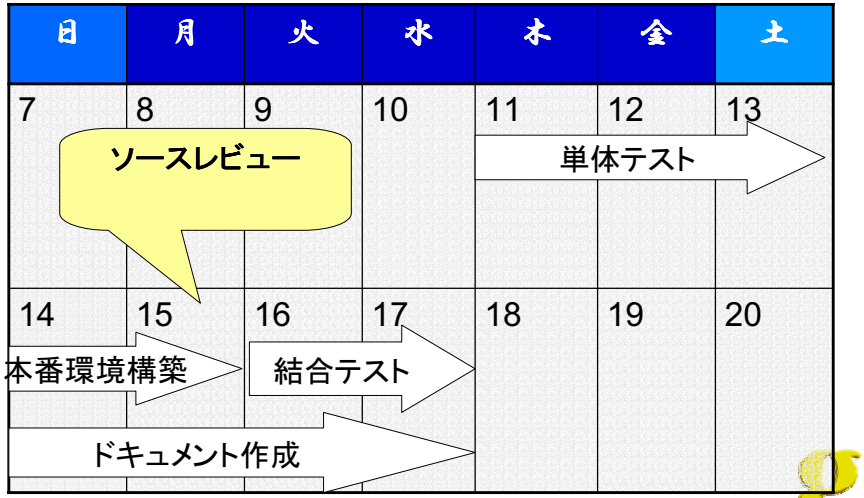
g-mod

12月22日以降の実績

- ソースコード精査
  - マージ（佐藤）
    - 各自が重複して変更を掛けたプログラムをマージした。
  - JavaDoc用コメント追加（各自）
    - JavaDoc用のコメントを追加した。
  - テストモジュール作成（各自）
    - 各自担当しているコーディング部分のテストモジュールを作成した。
  - 単体テスト（佐藤、松田）
    - 遠峰さんと合意が取れていないが、テストを実施した。
    - テスト仕様が少し古い為に実施できないテストもあった。
    - バグ件数0件
      →コーディングしながら動作確認をしていた為。
      →テスト不十分！テストケースの見直しが必要。

g-mod

2007/1/11～2007/1/17 予定



g-mod

作業時間実績

（）内は見積もり時間、単位は時間

- PM
  - 谷田部
    ・週報　0.5　・ML対応　0.5
    ・進捗報告　1.5
- 学生
  - 佐藤：13時間（16）
    ・コーディング　5(8)　・テストモジュール 2(4)
    ・環境構築　3(2.5)　・単体テスト　3（4）
  - 平澤：13時間(17.5)
    ・コーディング　6(8)　・テストモジュール 2(4)
    ・環境構築　3(2.5)　・ドキュメント修正 2(3)
  - 松田：12時間(14.5)
    ・コーディング　8(8)　・テストモジュール 2(4)
    ・環境構築　3(2.5)

```java
package main;

import geocode.RequestProcessor;

import java.io.File;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

import log.AccessLogger;
import log.LogWriter;
import log.SystemLogger;
import setting.GeocodeSettings;

/**
 * Geocoding Module
 *
 * @author g-mod Project
 * @version 1.0
 */
public class GeocodingModule {

  GeocodeSettings settings; // Geocoding Moduleの初期設定

  ServerSocket serverSocket; // Geocoding Moduleが利用するサーバソケット

  public static void main(String args[]) {
    GeocodingModule geocodingModule = new GeocodingModule();
    geocodingModule.run();
  }

  /**
   * Geocoding Moduleを起動する
   */
  public void run() {

    System.out.println("Geocoding Module Started");// 開始を知らせる
    initSettings();// モジュール設定を初期化する
    startLogger(); // ログの設定をする（これ以前にログに吐き出す必要が出た場合は、デフォルトのフォルダに出力する）
    startService();// サービスを開始する
  }

  /**
   * モジュールの設定を初期化する
   */
  private void initSettings() {
    settings = new GeocodeSettings();
    // 設定ファイルを読み込む
    if (settings.readConfigFile()) { // 読み込みに成功した場合
      // 設定ファイルが壊れているかチェックする
      if (settings.checkConfigFile() == false) { // 壊れている場合
        SystemLogger.writer.writeError(SystemLogger.ERROR_CFG_BROKEN, null);
        terminate();
      }
    } else { // 読み込みに失敗した場合
      SystemLogger.writer.writeError(SystemLogger.ERROR_CFG_CANT_READ, null);
      terminate();
    }
  }

  /**
   * サービスを開始する
   */
  private void startService() {
    try {
      SystemLogger.writer.writeInfo(SystemLogger.MODULE_START_SERVICE);
```

```
67        serverSocket = new ServerSocket(settings.getModulePort()); // サーバソケットを開く
68
69        // クライアントリクエストを処理する
70        while (true) {
71          // クライアントリクエストを待つ
72          Socket socket = serverSocket.accept();
73          // クライアントリクエストを受理するスレッドを作成する
74          try {
75            RequestProcessor thread = new RequestProcessor(socket,
76                settings);
77            thread.start();
78          } catch (Exception e) {
79            SystemLogger.writer.writeError(SystemLogger.ERROR_SOCKET, e);
80            terminate();
81          }
82        }
83      } catch (IOException e) {
84        SystemLogger.writer.writeError(SystemLogger.ERROR_SOCKET, e);
85        terminate();
86      }
87    }
88
89    /**
90     * ログ出力機能を起動する
91     */
92    private void startLogger() {
93      // ログの出力の有無を決める
94      // システムログの出力の有無を決める
95      if (settings.getLogLevel().equals("DEBUG")) {
96        LogWriter.setLogLevel(LogWriter.DEBUG);
97      } else if (settings.getLogLevel().equals("INFO")) {
98        LogWriter.setLogLevel(LogWriter.INFO);
99      } else if (settings.getLogLevel().equals("WARNING")){
100       LogWriter.setLogLevel(LogWriter.WARNING);
101     } else if (settings.getLogLevel().equals("ERROR")){
102       LogWriter.setLogLevel(LogWriter.ERROR);
103     }
104
105     // ログの出力先を決める
106     // システムログの出力先を決める
107     if ((new File(settings.getSystemLogPath())).isDirectory()) {
108       SystemLogger.writer.setPath(settings.getSystemLogPath());
109     }
110     // アクセスログの出力先を決める
111     if ((new File(settings.getAccessLogPath())).isDirectory()) {
112       AccessLogger.writer.setPath(settings.getAccessLogPath());
113     }
114
115     // エンコードを設定する
116     LogWriter.setEncode(settings.getLogEncode());
117   }
118
119   /**
120    * Geocoding Moduleを終了する
121    */
122   private void terminate() {
123     System.out.println("Geocoding Module Terminated");// 終了をコンソールに記す
124     System.exit(1);
125   }
126 }
```

```java
package geocode;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

import log.AccessLogger;
import log.SystemLogger;
import setting.GeocodeSettings;

/**
 * 1つのスレッドを表現するクラス
 *
 * @author Toshiyuki Sato
 *
 */
public class RequestProcessor extends Thread {

  private Socket socket = null;

  Geocoder geocoder;

  GeocodeSettings settings;

  /**
   * コンストラクタ
   *
   * @param socket
   * @param settings
   */
  public RequestProcessor(Socket socket, GeocodeSettings settings) {
    this.socket = socket;
    geocoder = new Geocoder(settings);
    this.settings = settings;
  }

  /**
   * リクエストに対応するレスポンスを返す
   */
  public void run() {
    try {
      // 前処理をする
      PrintWriter output = new PrintWriter(socket.getOutputStream(), true);// 出力用インタフェース
      BufferedReader input = new BufferedReader(new InputStreamReader(
          socket.getInputStream(), settings.getRequestEncode()));// 入力用インタフェース
      // BufferedReader input = new BufferedReader(new
      // InputStreamReader(socket.getInputStream(),
      // settings.getRequestEncode()));// 入力用インタフェース

      // レスポンスを返す
      String inputLine = input.readLine();// データを取得する
      AccessLogger.writer.writeInfo(AccessLogger.REQUEST_RECEIVE + " :"
          + inputLine);// ログに出力する
      SystemLogger.writer.writeDebug("Request recieved:" + inputLine
          + " from" + socket.getInetAddress());// デバッグプリント
      String result = geocoder.geocode(inputLine);// レスポンスする値を取得する
      SystemLogger.writer.writeDebug("Returning result:" + result);// デバッグプリント
      output.println(result);// 出力する
      SystemLogger.writer
          .writeDebug("Succeed returning result:" + result);// デバッグプリント

      // 後処理をする
      input.close(); // ストリームを閉じる
      output.close();
```

```
67        socket.close(); // ソケットを閉じる
68
69      } catch (IOException e) {
70        SystemLogger.writer.writeError(SystemLogger.ERROR_SOCKET, e);
71      }
72    }
73 }
```

```java
package geocode;

import java.util.List;

import log.AccessLogger;
import log.SystemLogger;
import setting.GeocodeSettings;

/**
 * 地名から緯度経度情報を取得するクラス
 *
 * @author Toshiyuki Sato
 *
 */
public class Geocoder {

  // 変数郡
  GeocodeSettings settings;

  /**
   * コンストラクタ
   *
   * @param settings
   */
  public Geocoder(GeocodeSettings settings) {
    this.settings = settings;
  }

  /**
   * 入力された住所からクライアントに返す値を用意する
   *
   * @param request
   * @return
   */
  public String geocode(String request) {
    // リクエストが不正の場合は終了する
    if (!checkRequest(request)) {// 不正の場合
      AccessLogger.writer.writeWarning(AccessLogger.WARNING_BAD_REQUEST);
      return null;
    }

    String result = getGeoData(request);// 地理情報を取得する
    // ログに出力する
    if (result != null) {// 地理情報が見つかった場合
      AccessLogger.writer.writeInfo(AccessLogger.REQUEST_FOUND + " :"
          + request);
    } else {// 地理情報が見つからなかった場合
      AccessLogger.writer.writeInfo(AccessLogger.REQUEST_NOTFOUND + " :"
          + request);
    }

    // 地理情報を返す
    return result;
  }

  /**
   * 出力用地理情報文字列を取得する
   *
   * @param request
   * @return
   */
  private String getGeoData(String request) {
    SystemLogger.writer.writeDebug("Try to get geodata");// デバッグプリント
    GeoDataManager manager = new GeoDataManager(settings);
    DBManager dbManager = new DBManager(settings);

    // 地理情報リストを取得する
```

```
68      manager.updateDatabase(request, dbManager);// DBを更新する
69      // int requestTimes = manager.getRequestTimes(request);// 回数を取得する
70      List<GeoData> geoDatas = manager.getGeoDatas(request, dbManager);
71      dbManager.closeConnection();// 接続を切る
72
73      // 地理情報リストから出力用文字列を取得する
74      String result = organizeGeocodedDatas(geoDatas);
75      SystemLogger.writer.writeDebug("Get geodata succeed:" + result);// デバッグプリント
76      return result;
77    }
78
79    /**
80     * 地理情報リストから出力用文字列を作成する
81     *
82     * @param geoDatas
83     * @return
84     */
85    private String organizeGeocodedDatas(List<GeoData> geoDatas) {
86      if (geoDatas == null || geoDatas.isEmpty()) {// リストが空の場合
87        // TODO:地理情報を取得できませんでしたログを吐く（エラーではない）
88        return null;
89      }
90
91      // リストに地理情報がある場合、出力用文字列を作成する
92      String result = null;
93      for (GeoData data : geoDatas) {
94        if (data.isNotEmpty()) {
95          if (result == null){//一つ目の要素の場合
96            result = "";
97          }else {// 2つ目以降の要素の場合
98            // セミコロンを挿入する
99            result += ";";
100         }
101         // 「緯度,経度」の文字列を作成する
102         result += data.getLatitude() + "," + data.getLongitude();
103       }
104     }
105     return result;
106   }
107
108   /**
109    * リクエストの正当性をチェックする
110    *
111    * @param request
112    *            リクエスト
113    * @return リクエストが正当な場合trueを、不正な場合falseを返す
114    */
115   private boolean checkRequest(String request) {
116     return request != null && !(request.equals(""));
117   }
118 }
```

```java
package geocode;

import java.util.List;

import log.SystemLogger;

import setting.GeocodeSettings;

/**
 * 地理情報データとやり取りするクラス
 *
 * @author Toshiyuki Sato
 *
 */
public class GeoDataManager {

  // 変数郡
  GeocodeSettings settings;

  /**
   * コンストラクタ
   *
   * @param settings
   */
  public GeoDataManager(GeocodeSettings settings) {
    this.settings = settings;
  }

  /**
   * データベースを更新する
   *
   * @param searchKey
   */
  public void updateDatabase(String searchKey, DBManager manager) {
    SystemLogger.writer.writeDebug("Try to update DB for " + searchKey);// デバッグプリント
    updateSearchKey(searchKey, manager);
    updateData(searchKey, manager);
    SystemLogger.writer
        .writeDebug("Finished to update DB for " + searchKey);// デバッグプリント
  }

  /**
   * 地理情報を取得する
   *
   * @param searchKey
   * @return
   */
  public List<GeoData> getGeoDatas(String searchKey, DBManager manager) {
    SystemLogger.writer.writeDebug("Try to get GeoDatas from DB for"
        + searchKey);// デバッグプリント
    List<GeoData> geoData = manager.getGeodata(searchKey);
    SystemLogger.writer.writeDebug("got" + geoData.size()
        + " GeoDatas from DB");// デバッグプリント
    return geoData;
  }

  /**
   * 検索キーを更新する
   *
   * @param searchKey
   */
  private void updateSearchKey(String searchKey, DBManager dbManager) {
    SystemLogger.writer.writeDebug("Try to update SearchKey Table for "
        + searchKey);// デバッグプリント
    if (dbManager.keyExist(searchKey)) {// 検索キーがDBに存在している場合
      dbManager.updateRequestTimes(searchKey);// アクセスの回数を増やす
      if (dbManager.keyExpired(searchKey)) {// 検索キーの有効期間が過ぎている場合
```

```
68          // DBを更新する
69          APIManager apiManager = new APIManager(settings);
70          List<String> searchList = apiManager.getSearchList(searchKey);
71          dbManager.updateKey(searchKey, searchList);
72        } else {
73          SystemLogger.writer
74              .writeDebug("No need to update SearchKey for "
75                  + searchKey);// デバッグプリント
76        }
77      } else {// 存在していない場合
78        // 新規に取得し、DBに追加する
79        APIManager apiManager = new APIManager(settings);
80        List<String> searchList = apiManager.getSearchList(searchKey);
81        dbManager.addKey(searchKey, searchList);
82      }
83      SystemLogger.writer.writeDebug("Finished update SearchKey Table for "
84          + searchKey);// デバッグプリント
85    }
86
87    /**
88     * 地名を更新する
89     *
90     * @param searchKey
91     */
92    private void updateData(String searchKey, DBManager dbManager) {
93      SystemLogger.writer.writeDebug("Try to update GeoData Table");// デバッグプリント
94      List<String> locationNames = dbManager.getLocationNameList(searchKey);
95      // 地名ごとに地理情報を更新する
96      if (locationNames != null && !locationNames.isEmpty()) {// 地名が存在する場合
97        for (String locationName : locationNames) {
98          SystemLogger.writer.writeDebug("Target location name :"
99              + locationName);// デバッグプリント
100         if (dbManager.dataExist(locationName)) {// 地理情報がDBに存在している場合
101           if (dbManager.dataExpired(locationName)) {// 地理情報の有効期間が過ぎている場合
102             // DBを更新する
103             APIManager apiManager = new APIManager(settings);
104             GeoData geoData = apiManager.getLocation(locationName);
105             if (geoData.isNotEmpty()) {
106               dbManager.updateData(geoData);
107             }
108           } else {// 過ぎていない場合
109             SystemLogger.writer
110                 .writeDebug("No need to update location :"
111                     + locationName);// デバッグプリント
112           }
113         } else {// 存在していない場合
114           // 新規に取得し、DBに追加する
115           APIManager apiManager = new APIManager(settings);
116           GeoData geoData = apiManager.getLocation(locationName);
117           dbManager.addData(geoData);
118         }
119       }
120     } else {
121       SystemLogger.writer.writeDebug("No target to update geodata");// デバッグプリント
122     }
123     SystemLogger.writer.writeDebug("Finished update GeoData Table");// デバッグプリント
124   }
125
126 }
```

```java
package geocode;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import log.SystemLogger;
import setting.GeocodeSettings;

public class DBManager {

  private String dbURL; // データベースの接続先

  private String dbUserName; // データベースのユーザ名

  private String dbPass; // データベースのパスワード

  private GeocodeSettings settings; // 設定ファイルの情報

  private long requestExpiration; // リクエストの有効期限

  private long dataExpiration; // 地理情報の有効期限

  private int returnMax; // 返す地名リストの長さ

  private Connection con;

  private Statement stat;

  /**
   * コンストラクタ
   */
  public DBManager(GeocodeSettings settings) {
    this.settings = settings;
    initSettings();
    openConnection();
  }

  /**
   * 設定を初期化する
   */
  void initSettings() {
    dbURL = settings.getDBURL();
    dbUserName = settings.getDBUserName();
    dbPass = settings.getDBPass();
    requestExpiration = settings.getSearchKeyExpiration() * 1000L * 60L
        * 60L * 24L;
    dataExpiration = settings.getCacheExpiration() * 1000L * 60L * 60L
        * 24L;
    returnMax = settings.getReturnMax();
    if (dbPass == null) {
      dbPass = "";
    }
  }

  /**
   * データベースに接続する
   *
   * @return 成功すればtrueを、失敗すればfalseを返す
   */
  private boolean openConnection() {
    try {
```

```
68        Class.forName("org.gjt.mm.mysql.Driver");
69        con = DriverManager.getConnection(dbURL, dbUserName, dbPass);
70        stat = con.createStatement();
71        return true;
72      } catch (ClassNotFoundException e) {
73        SystemLogger.writer.writeError(SystemLogger.ERROR_DB_CONNECT_FAIL,
74            e);
75        terminate();
76      } catch (Exception e) {
77        SystemLogger.writer.writeError(SystemLogger.ERROR_DB_CONNECT_FAIL,
78            e);
79        terminate();
80      }
81      return false;
82    }
83
84    /**
85     * データベースの接続をきる
86     *
87     * @return 成功すればtrueを、失敗すればfalseを返す
88     */
89    public boolean closeConnection() {
90      try {
91        stat.close();
92        con.close();
93        return true;
94      } catch (Exception e) {
95        SystemLogger.writer.writeError(
96            SystemLogger.ERROR_DB_DISCONNECT_FAIL, e);
97      }
98      return false;
99    }
100
101   public int getRequestTimes(String searchKey) {
102     int requestTimes = 0;
103     try {
104       // サーチキーのリクエスト回数を増やす
105       String sql = "SELECT requestTimes FROM geocodingrequest "
106           + "WHERE searchQuery = '" + searchKey + "'";
107       ResultSet rs = stat.executeQuery(sql);
108       while (rs.next())
109         requestTimes = rs.getInt("requestTimes");
110
111     } catch (Exception e) {
112       SystemLogger.writer.writeError(SystemLogger.ERROR_DB_CONNECT_FAIL,
113           e);
114       terminate();
115     }
116     return requestTimes;
117   }
118
119   /**
120    * 検索キーがデータベースに存在するかどうか確かめる
121    *
122    * @param key
123    *            検索キー
124    * @return キーに対応するレコードがデータベースにあるかどうか調べる。
125    */
126   public boolean keyExist(String key) {
127     try {
128       String sql = "SELECT * FROM geocodingrequest "
129           + "WHERE searchQuery = '" + key + "'";
130       ResultSet rs = stat.executeQuery(sql);
131       while (rs.next())
132         return true;
133     } catch (Exception e) {
134       SystemLogger.writer.writeError(SystemLogger.ERROR_DB_CONNECT_FAIL,
```

```
135           e);
136       terminate();
137     }
138     return false;
139   }
140
141   /**
142    * サーチキーの有効期限を確かめる
143    *
144    * @param key
145    *            サーチキー
146    * @return 期限切れの場合trueを、期限内の場合falseを返す
147    */
148   public boolean keyExpired(String key) {
149     try {
150       String sql = "SELECT lastUpdatedRequest FROM geocodingrequest "
151           + "WHERE searchQuery = '" + key + "'";
152       ResultSet rs = stat.executeQuery(sql);
153       while (rs.next()) {
154         // 現在の日付を取得する
155         Date dateNow = new Date();
156         // 最終更新日を取得する
157         Date date = rs.getDate("lastUpdatedRequest");
158         long timeDiff = dateNow.getTime() - date.getTime();
159         // 有効期限を過ぎているかどうか確かめる
160         if (timeDiff > requestExpiration)
161           return true; // 有効期限を過ぎている
162       }
163     } catch (Exception e) {
164       SystemLogger.writer.writeError(SystemLogger.ERROR_DB_CONNECT_FAIL,
165           e);
166       terminate();
167     }
168     return false; // 有効期限を過ぎてない
169   }
170
171   /**
172    * サーチキーをデータベースに登録する
173    *
174    * @param searchKey
175    *            サーチキー
176    * @param keyList
177    *            地名リスト
178    */
179   public void addKey(String searchKey, List<String> keyList) {
180     try {
181       SystemLogger.writer.writeDebug("Adding key to RT :" + searchKey);// デバッグプリント
182       java.sql.Date date = new java.sql.Date((new Date().getTime()));
183       String sql = "insert into geocodingrequest " + "values ('"
184           + searchKey + "',1,'" + date + "')";
185       stat.executeUpdate(sql);
186       int i = 0;
187       int rank = 1;
188       if (keyList != null) {
189         while (i < keyList.size()) {
190           String locationName = keyList.get(i);
191           SystemLogger.writer.writeDebug("Adding key to LT :"
192               + searchKey + "," + locationName);// デバッグプリント
193           sql = "insert into geocodinglink values ('" + searchKey
194               + "','" + locationName + "'," + rank + ")";
195           stat.executeUpdate(sql);
196           i++;
197           rank++;
198         }
199       }
200     } catch (SQLException e) { // 主キーの重ね入れをした場合
201       SystemLogger.writer
```

```
202           .writeWarning(SystemLogger.WARNING_DB_INSERT_FAIL);
203       } catch (Exception e) {
204         SystemLogger.writer.writeError(SystemLogger.ERROR_DB_CONNECT_FAIL,
205             e);
206         terminate();
207       }
208     }
209
210     /**
211      * リクエストの回数を更新する
212      *
213      * @param searchKey
214      *            サーチキー
215      */
216     public void updateRequestTimes(String searchKey) {
217       try {
218         // サーチキーのリクエスト回数を増やす
219         String sql = "SELECT requestTimes FROM geocodingrequest "
220             + "WHERE searchQuery = '" + searchKey + "'";
221         ResultSet rs = stat.executeQuery(sql);
222         int requestTimes = 0;
223         while (rs.next())
224           requestTimes = rs.getInt("requestTimes");
225         if (requestTimes < Integer.MAX_VALUE)
226           requestTimes++;
227         sql = "update geocodingrequest set requestTimes = " + requestTimes
228             + " where searchQuery = '" + searchKey + "'";
229         stat.executeUpdate(sql);
230       } catch (Exception e) {
231         SystemLogger.writer.writeError(SystemLogger.ERROR_DB_CONNECT_FAIL,
232             e);
233         terminate();
234       }
235     }
236
237     /**
238      * サーチキーの情報を更新する
239      *
240      * @param searchKey
241      *            サーチキー
242      * @param keyList
243      *            地名リスト
244      */
245     public void updateKey(String searchKey, List<String> keyList) {
246       try {
247         // サーチキーの有効期限を確認する
248         if (keyExpired(searchKey)) { // 有効期限が切れている場合
249           SystemLogger.writer.writeDebug("Updating key to LT :"
250               + searchKey + "," + searchKey);// デバッグプリント
251           // 有効期限を更新する
252           java.sql.Date date = new java.sql.Date((new Date().getTime()));
253           String sql = "update geocodingrequest set lastUpdatedRequest = '"
254               + date + "' where searchQuery = '" + searchKey + "'";
255           stat.executeUpdate(sql);
256
257           // サーチキーに該当する地名リストを更新する
258           sql = "DELETE FROM geocodinglink WHERE searchQuery = '"
259               + searchKey + "'";
260           stat.executeUpdate(sql);
261           int i = 0;
262           if (keyList != null) {
263             while (i < keyList.size()) {
264               String locationName = keyList.get(i);
265               SystemLogger.writer.writeDebug("Adding key to LT :"
266                   + searchKey + "," + locationName);// デバッグプリント
267               int rank = i + 1;
268               sql = "insert into geocodinglink " + "values ('"
```

```
269                 + searchKey + "','" + locationName + "',"
270                 + rank + ")";
271             stat.executeUpdate(sql);
272             i++;
273           }
274         }
275       }
276     } catch (Exception e) {
277       SystemLogger.writer.writeError(SystemLogger.ERROR_DB_CONNECT_FAIL,
278           null);
279       terminate();
280     }
281   }
282
283   /**
284    * 地名に該当する地理情報がデータベースに存在するかどうか確かめる
285    *
286    * @param key
287    *            地名
288    * @return ある場合trueを、ない場合falseを返す
289    */
290   public boolean dataExist(String key) {
291     try {
292       String sql = "SELECT * FROM geocodingdata "
293           + "WHERE locationName = '" + key + "'";
294       ResultSet rs = stat.executeQuery(sql);
295       while (rs.next())
296         return true;
297     } catch (Exception e) {
298       SystemLogger.writer.writeError(SystemLogger.ERROR_DB_CONNECT_FAIL,
299           e);
300       terminate();
301     }
302     return false;
303   }
304
305   /**
306    * 地理情報の有効期限を確かめる
307    *
308    * @param key
309    *            地名
310    * @return 期限切れの場合trueを、期限内の場合falseを返す
311    */
312   public boolean dataExpired(String key) {
313     try {
314       String sql = "SELECT lastUpdated FROM geocodingdata "
315           + "WHERE locationName = '" + key + "'";
316       ResultSet rs = stat.executeQuery(sql);
317       while (rs.next()) {
318         // 現在の日付を取得する
319         Date dateNow = new Date();
320         // 最終更新日を取得する
321         Date date = rs.getDate("lastUpdated");
322         long timeDiff = dateNow.getTime() - date.getTime();
323         // 有効期限を過ぎているかどうか確かめる
324         if (timeDiff > dataExpiration)
325           return true; // 有効期限を過ぎている
326       }
327     } catch (Exception e) {
328       SystemLogger.writer.writeError(SystemLogger.ERROR_DB_CONNECT_FAIL,
329           e);
330       terminate();
331     }
332     return false; // 有効期限を過ぎてない
333   }
334
335   /**
```

```
336     * データベースに地理情報を追加する
337     *
338     * @param data
339     *            地理情報
340     */
341    public void addData(GeoData data) {
342      try {
343
344        SystemLogger.writer.writeDebug("Adding data to DT :"
345            + data.getLocationName() + "," + data.getLatitude() + ","
346            + data.getLongitude());// デバッグプリント
347        java.sql.Date date = new java.sql.Date(new Date().getTime());
348        String sql = "insert into geocodingdata
    (locationName,latitude,longitude,insertDate,lastUpdated) "
349            + "values('"
350            + data.getLocationName()
351            + "','"
352            + data.getLatitude()
353            + "','"
354            + data.getLongitude()
355            + "','"
356            + date + "','" + date + "')";
357        stat.executeUpdate(sql);
358      } catch (SQLException e) { // 主キーの重ね入れをした場合
359        SystemLogger.writer
360            .writeWarning(SystemLogger.WARNING_DB_INSERT_FAIL);
361      } catch (Exception e) {
362        SystemLogger.writer.writeError(SystemLogger.ERROR_DB_CONNECT_FAIL,
363            e);
364        terminate();
365      }
366    }
367
368    /**
369     * 地理情報を更新する
370     *
371     * @param data
372     *            地理情報
373     */
374    public void updateData(GeoData data) {
375      try {
376        SystemLogger.writer.writeDebug("Updating data to DT :"
377            + data.getLocationName() + "," + data.getLatitude() + ","
378            + data.getLongitude());// デバッグプリント
379        java.sql.Date date = new java.sql.Date(new Date().getTime());
380        String sql = "update geocodingdata set latitude = "
381            + data.getLatitude() + ", longitude = "
382            + data.getLongitude() + ", lastUpdated = '" + date
383            + "' where locationName = '" + data.getLocationName() + "'";
384        stat.executeUpdate(sql);
385      } catch (Exception e) {
386        SystemLogger.writer.writeError(SystemLogger.ERROR_DB_CONNECT_FAIL,
387            e);
388        terminate();
389      }
390    }
391
392    /**
393     * サーチキーから地名リストを取得する
394     *
395     * @param key
396     *            サーチキー
397     * @return 地名リスト
398     */
399    public List<String> getLocationNameList(String key) {
400      List<String> answerList = new ArrayList<String>();
401      try {
```

```
402       String sql = "SELECT locationName FROM geocodinglink "
403           + "WHERE searchQuery = '" + key + "'";
404       ResultSet rs = stat.executeQuery(sql);
405       int num = 0;
406       while (rs.next() && num < returnMax) {
407         answerList.add(rs.getString("locationName"));
408         num++;
409       }
410     } catch (Exception e) {
411       SystemLogger.writer.writeError(SystemLogger.ERROR_DB_CONNECT_FAIL,
412           e);
413       terminate();
414     }
415     return answerList;
416   }
417
418   /**
419    * サーチキーから地理情報を取得する
420    *
421    * @param key
422    * @return 地理情報リスト
423    */
424   public List<GeoData> getGeodata(String key) {
425     List<GeoData> answerList = new ArrayList<GeoData>();
426     try {
427       String sql = "SELECT geocodingdata.locationName, geocodingdata.latitude,
    geocodingdata.longitude "
428           + "FROM geocodingrequest, geocodinglink, geocodingdata "
429           + "WHERE geocodingrequest.searchQuery = geocodinglink.searchQuery "
430           + "AND geocodinglink.locationName = geocodingdata.locationName "
431           + "AND geocodingrequest.searchQuery = '" + key + "';";
432       ResultSet rs = stat.executeQuery(sql);
433       int num = 0;
434       while (rs.next() && num < returnMax) {
435         GeoData data = new GeoData(rs.getString("locationName"), rs
436             .getString("latitude"), rs.getString("longitude"));
437         answerList.add(data);
438         num++;
439       }
440     } catch (Exception e) {
441       SystemLogger.writer.writeError(SystemLogger.ERROR_DB_CONNECT_FAIL,
442           e);
443       terminate();
444     }
445     return answerList;
446   }
447
448   /**
449    * Geocoding Moduleを終了する
450    */
451   private void terminate() {
452     System.out.println("Geocoding Module Terminated");// 終了をコンソールに表示する
453     closeConnection();
454     System.exit(1);
455   }
456
457 }
```

```java
package geocode;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLEncoder;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import log.SystemLogger;
import setting.GeocodeSettings;

public class APIManager {
  // 定数郡
  public static final String NOTFOUND = "null";// データが取得できなかった場合のヌル文字

  // 設定用変数郡
  private String apiAddress = "http://www.geocoding.jp/api/?q=";

  private String encode = "UTF-8";

  private int waitTimeForNextConnection = 5000; // API接続への待ち時間

  // 変数郡
  static Date lastAccess = new Date(0);

  static GeoData tempGeoData = null;// 地名取得時の保存用

  /**
   * コンストラクタ
   *
   * @param dBConnecter
   * @param settings
   */
  public APIManager(GeocodeSettings settings) {
    initialize(settings);
  }

  /**
   * 設定の初期化をする
   *
   * @param settings
   */
  private void initialize(GeocodeSettings settings) {
    apiAddress = settings.getApiURL();
    encode = settings.getApiEncode();
    waitTimeForNextConnection = settings.getRequestWaitTime();
  }

  /**
   * 検索キーから地名リストを取得するメソッド
   *
   * @param searchKey
   * @return
   */
  public List<String> getSearchList(String searchKey) {
    SystemLogger.writer.writeDebug("Access API to get Location names");// デバッグプリント
    try {
      waitToAccessAPI();// 接続する
      // リソースを指定する
      URL url = new URL(apiAddress + URLEncoder.encode(searchKey, encode));
      SystemLogger.writer.writeDebug("Connect to :"
          + url.toExternalForm());

      // ストリームを生成する
```

```
68        BufferedReader in = new BufferedReader(new InputStreamReader(url
69            .openStream(), encode));
70        String line;
71        List<String> getLines = new ArrayList<String>();
72        while ((line = in.readLine()) != null) {
73          getLines.add(line);
74        }
75
76        // 地名が一つしかない場合
77        if (isResponseSingle(getLines)) {
78          tempGeoData = getGeocodedDataFromResponse(getLines);// 保存しておく
79          return getLocationNameFromResponse(getLines);
80        }
81        // 地名が複数ある場合
82        else if (isResponseMultiple(getLines)) {
83          return getLocationNamesFromResponse(getLines);
84        }
85        // 地名がない場合
86        else {
87          return null;
88        }
89      } catch (IOException e) {
90        SystemLogger.writer.writeError(SystemLogger.ERROR_SOCKET, e);
91      }
92      return null;
93    }
94
95    /**
96     * 地名から地理情報を取得するメソッド
97     *
98     * @param location
99     * @return
100    */
101   public GeoData getLocation(String location) {
102     GeoData geoData = new GeoData(location, null, null);
103     // 以前に取得し、保存されている場合はそこから取得する
104     if (tempGeoData != null
105         && tempGeoData.getLocationName().equals(location)) {
106       geoData = tempGeoData;
107       tempGeoData = null;
108       SystemLogger.writer.writeDebug("Got GeoData before access API:"
109           + geoData.getLocationName() + "," + geoData.getLatitude()
110           + "," + geoData.getLongitude());// デバッグプリント
111       return geoData;
112     } else {// 異なった場合（更新期限切れの場合等）
113       tempGeoData = null;
114     }
115
116     // 地名から地理情報を取得する
117     try {
118       SystemLogger.writer.writeDebug("Access API to get GeoData");// デバッグプリント
119       waitToAccessAPI();// 接続する
120       // リソースを指定する
121       URL url = new URL(apiAddress + URLEncoder.encode(location, encode));
122       SystemLogger.writer.writeDebug("Connect to :"
123           + url.toExternalForm());
124       // ストリームを生成する
125       BufferedReader in = new BufferedReader(new InputStreamReader(url
126           .openStream(), encode));
127       // 最後に接続した時間を記憶する
128       lastAccess = new Date();
129
130       // レスポンスをリストとして保持する
131       String line;
132       List<String> getLines = new ArrayList<String>();
133       while ((line = in.readLine()) != null) {
134         getLines.add(line);
```

```
135       }
136       // 取得した値に記載された地理情報の数によって処理を変える
137       if (isResponseSingle(getLines)) {// 単独の場合
138         geoData = getGeocodedDataFromResponse(getLines);// 取得した値から地理情報を作成する
139         if (geoData != null) {
140         }
141       } else {// 取得できなかった場合
142         geoData = new GeoData(location, null, null);
143       }
144
145       // ストリームを閉じる
146       in.close();
147
148     } catch (IOException e) {
149       SystemLogger.writer.writeError(SystemLogger.ERROR_SOCKET, e);
150     }
151     SystemLogger.writer.writeDebug("Got GeoData from API:"
152         + geoData.getLocationName() + "," + geoData.getLatitude() + ","
153         + geoData.getLongitude());// デバッグプリント
154     return geoData;
155   }
156
157   /**
158    * API接続前に5秒間待つ
159    *
160    */
161   private void waitToAccessAPI() {
162     // 前回のアクセスとの差分を取得する
163     long timeDifference = new Date().getTime() - lastAccess.getTime();
164     // 待ち時間が必要かどうか判定する
165     if (timeDifference < waitTimeForNextConnection) {// 必要な場合
166       long timeToWait = waitTimeForNextConnection - timeDifference;
167       try {
168         Thread.sleep(timeToWait);
169       } catch (InterruptedException e) {
170         SystemLogger.writer.writeError(SystemLogger.ERROR_SOCKET, e);
171       }
172     }
173   }
174
175   /**
176    * 検索結果が単独回答のものかどうかを判定する
177    *
178    * @param lines
179    * @return
180    */
181   private boolean isResponseSingle(List<String> lines) {
182     for (String string : lines) {
183       // 複数の選択肢がある場合
184       if (string.startsWith("<choices>")) {
185         return false;
186       }
187       // 緯度経度情報を返している場合
188       else if (string.startsWith("<lat>")) {
189         return true;
190       }
191     }
192     return false;
193   }
194
195   /**
196    * 検索結果が複数回答のものかどうかを判定する
197    *
198    * @param lines
199    * @return
200    */
201   private boolean isResponseMultiple(List<String> lines) {
```

```
202     for (String string : lines) {
203       // 複数の選択肢がある場合
204       if (string.startsWith("<choices>")) {
205         return true;
206       }
207       // 例外の場合
208       else if (string.startsWith("<lat>")) {
209         return false;
210       }
211     }
212     return false;
213   }
214
215   private GeoData getGeocodedDataFromResponse(List<String> getLines) {
216     String latitude = "";
217     String longitude = "";
218     String locationName = "";
219     // 地理情報を取得する
220     for (String string : getLines) {
221       // 単独の選択肢の場合
222       if (string.startsWith("<lat>")) {
223         latitude = getLatitude(string);
224       } else if (string.startsWith("<lng>")) {
225         longitude = getLongitude(string);
226       } else if (string.startsWith("<address>")) {
227         locationName = getAddress(string);
228       }
229     }
230     // 処理する
231     if (latitude.equals("") || longitude.equals("")) {
232       latitude = NOTFOUND;
233       longitude = NOTFOUND;
234     }
235     GeoData geoData = new GeoData(locationName, latitude, longitude);
236     return geoData;
237   }
238
239   /**
240    * 緯度の行から緯度だけを返す
241    *
242    * @param lat
243    * @return
244    */
245   private String getLatitude(String lat) {
246     String newString = lat.replace("<lat>", "");
247     newString = newString.replace("</lat>", "");
248     return newString;
249   }
250
251   /**
252    * 軽度の行から緯度だけを返す
253    *
254    * @param lat
255    * @return
256    */
257   private String getLongitude(String lng) {
258     String newString = lng.replace("<lng>", "");
259     newString = newString.replace("</lng>", "");
260     return newString;
261   }
262
263   /**
264    * 地理名を返す
265    *
266    * @param input
267    * @return
268    */
```

```
269   private String getAddress(String input) {
270     String newString = input.replace("<address>", "");
271     newString = newString.replace("</address>", "");
272     return newString;
273   }
274
275   /**
276    * 選択肢の中から一つの選択肢を取り出す
277    *
278    * @param choice
279    * @return
280    */
281   private String getOneChoice(String choice) {
282     String newString = choice.replace("<choice>", "");
283     newString = newString.replace("</choice>", "");
284     return newString;
285   }
286
287   /**
288    * レスポンスから地名を取得する
289    *
290    * @param getLines
291    * @return
292    */
293   private List<String> getLocationNameFromResponse(List<String> getLines) {
294     List<String> locationNames = new ArrayList<String>();
295     for (String string : getLines) {
296       if (string.startsWith("<address>")) {
297         locationNames.add(getAddress(string));
298         SystemLogger.writer.writeDebug("got location:"
299             + getAddress(string));// デバッグ出力
300         return locationNames;
301       }
302     }
303     SystemLogger.writer.writeDebug("no location found");// デバッグ出力
304     return null;
305   }
306
307   /**
308    * レスポンスから地名一覧を取得する
309    *
310    * @param getLines
311    * @return
312    */
313   private List<String> getLocationNamesFromResponse(List<String> getLines) {
314     List<String> locationNames = new ArrayList<String>();
315     for (String string : getLines) {
316       // 複数の選択肢がある場合
317       if (string.startsWith("<choice>")) {
318         locationNames.add(getOneChoice(string));
319         SystemLogger.writer.writeDebug("got location:"
320             + getOneChoice(string));// デバッグ出力
321       }
322     }
323     return locationNames;
324   }
325
326 }
```

```java
package geocode;


/**
 * 地理情報を表現するクラス
 *
 * @author Toshiyuki Sato
 *
 */
public class GeoData {

  private String locationName;

  private String latitude;

  private String longitude;

  /**
   * コンストラクタ
   *
   * @param locationName
   * @param latitude
   * @param longitude
   */
  public GeoData(String locationName, String latitude, String longitude) {
    this.locationName = locationName;
    this.latitude = latitude;
    this.longitude = longitude;
  }

  /**
   * 地理情報が有効かどうかを判定する
   *
   * @return
   */
  public boolean isNotEmpty() {
    return this.latitude != null && this.longitude != null
        && this.locationName != null && !latitude.equals(APIManager.NOTFOUND)
        && !longitude.equals(APIManager.NOTFOUND);
  }

  /*****************************************************************************
   * ゲッター・セッター
   *****************************************************************************/

  public String getLatitude() {
    return latitude;
  }

  public void setLatitude(String latitude) {
    this.latitude = latitude;
  }

  public String getLocationName() {
    return locationName;
  }

  public void setLocationName(String locationName) {
    this.locationName = locationName;
  }

  public String getLongitude() {
    return longitude;
  }

  public void setLongitude(String longitude) {
    this.longitude = longitude;
```

```
68     }
69
70 }
```

```
 1 package log;
 2
 3 import java.io.BufferedWriter;
 4 import java.io.File;
 5 import java.io.FileOutputStream;
 6 import java.io.IOException;
 7 import java.io.OutputStreamWriter;
 8 import java.text.SimpleDateFormat;
 9 import java.util.Date;
10
11 /**
12  * ログの出力を管理するクラス
13  *
14  * @author Hideyuki Hirasawa
15  * @version 0.1
16  */
17 public class LogWriter {
18
19   public static final int ERROR = 0;
20
21   public static final int WARNING = 1;
22
23   public static final int INFO = 2;
24
25   public static final int DEBUG = 3;
26
27   private static int logLevel = DEBUG;
28
29   private static String encode = "UTF-8";
30
31   private String path = ".";
32
33   private String filename = "default";
34
35   private String extension = ".txt";
36
37   /**
38    * コンストラクタ
39    *
40    * @param filename
41    *            ファイル名
42    * @param extension
43    *            ファイル拡張子
44    */
45   public LogWriter(String filename, String extension) {
46     this.filename = filename;
47     this.extension = extension;
48   }
49
50   /**
51    * 出力エンコードを変更する
52    *
53    * @param newEncode
54    *            新しい文字エンコード
55    */
56   public static void setEncode(String encode) {
57     LogWriter.encode = encode;
58   }
59
60   /**
61    * ログレベルを設定する
62    *
63    * @param logLevel
64    *            新しいログレベル
65    */
66   public static void setLogLevel(int logLevel) {
67     LogWriter.logLevel = logLevel;
```

```
 68   }
 69
 70   /**
 71    * 出力パスを設定する
 72    *
 73    * @param newPath
 74    */
 75   public void setPath(String newPath) {
 76     path = newPath;
 77   }
 78
 79   /**
 80    * DEBUGメッセージをログに出力する
 81    *
 82    * @param message
 83    *            メッセージ
 84    */
 85   public void writeDebug(String message) {
 86     if (logLevel >= DEBUG) {
 87       writeLog(getFilePath(), "DEBUG: " + message);
 88     }
 89   }
 90
 91   /**
 92    * INFOメッセージをログに出力する
 93    *
 94    * @param message
 95    *            メッセージ
 96    */
 97   public void writeInfo(String message) {
 98     if (logLevel >= INFO) {
 99       writeLog(getFilePath(), "INFO: " + message);
100     }
101   }
102
103   /**
104    * WARNINGメッセージをログに出力する
105    *
106    * @param message
107    *            メッセージ
108    */
109   public void writeWarning(String message) {
110     if (logLevel >= WARNING) {
111       writeLog(getFilePath(), "WARNING: " + message);
112     }
113   }
114
115   /**
116    * ERRORメッセージをログに出力
117    *
118    * @param message
119    *            メッセージ
120    */
121   public void writeError(String message) {
122     if (logLevel >= ERROR) {
123       writeLog(getFilePath(), "ERROR: " + message);
124     }
125   }
126
127   /**
128    * ERRORメッセージをログに出力
129    *
130    * @param message
131    *            メッセージ
132    * @param e
133    */
134   public void writeError(String message, Exception e) {
```

```
135     if (logLevel >= ERROR) {
136       writeLog(getFilePath(), "ERROR: " + message, e);
137     }
138   }
139
140   /**
141    * ログを出力する
142    *
143    * @param path
144    *            パス名
145    * @param content
146    *            出力内容
147    */
148   public void writeLog(String filePath, String content) {
149     writeLog(filePath, content, null);
150   }
151
152   /**
153    * ログを出力する
154    *
155    * @param path
156    *            パス名
157    * @param content
158    *            出力内容
159    */
160   public void writeLog(String filePath, String content, Exception ec) {
161     try {
162       File log = new File(filePath);
163       synchronized (log) {
164         FileOutputStream fOut = new FileOutputStream(filePath, true);
165         OutputStreamWriter oStream = new OutputStreamWriter(fOut,
166             encode);
167         BufferedWriter bWriter = new BufferedWriter(oStream);
168         bWriter.write(getDateTime() + content);
169         bWriter.newLine();
170         if (ec != null && ec.getMessage() != null) {
171           bWriter.write(ec.getMessage());
172           bWriter.newLine();
173         }
174         bWriter.close();
175       }
176     } catch (IOException e) {
177     }
178   }
179
180   /**
181    * ログのパス名を返す
182    *
183    * @return
184    */
185   public String getFilePath() {
186     return path + File.separator + filename + getDate() + extension;
187   }
188
189   /**
190    * 日付を取得する
191    *
192    * @return 出力例： "20070101"  (2007年1月1日)
193    */
194   private String getDate() {
195     SimpleDateFormat df = new SimpleDateFormat("yyyyMMdd");
196     return df.format(new Date());
197   }
198
199   /**
200    * 日付と時間を取得する
201    *
```

```
202     * @return 出力例： "2007/01/01 00:00:00"  (2007年1月1日 0時0分0秒)
203     */
204    private String getDateTime() {
205      SimpleDateFormat df = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss ");
206      return df.format(new Date());
207    }
208
209 }
```

```
 1 package log;
 2
 3
 4 /**
 5  * アクセスログを管理するクラス
 6  *
 7  * @author Hideyuki Hirasawa
 8  * @version 0.1
 9  */
10 public class AccessLogger{
11
12   public static final String ACCESS_LOG = "accessLog_"; // アクセスログ名
13
14   public static final String EXT = ".txt"; // ログの拡張子
15
16   public static final LogWriter writer = new LogWriter(ACCESS_LOG, EXT);
17
18   public static final String REQUEST_RECEIVE = "INFO received request ";
19
20   public static final String REQUEST_CONTENT = "INFO request: ";
21
22   public static final String REQUEST_FOUND = "INFO found entry(s) for ";
23
24   public static final String REQUEST_NOTFOUND = "INFO no entry(s) for ";
25
26   public static final String WARNING_BAD_REQUEST = "WARNING bad request";
27
28 }
```

```java
package log;

/**
 * モジュールログを管理するクラス
 *
 * @author Hideyuki Hirasawa
 * @version 0.1
 */
public class SystemLogger {

  public static final String SYSTEM_LOG = "systemLog_"; // アクセスログ名

  public static final String EXT = ".txt"; // ログの拡張子

  public static final LogWriter writer = new LogWriter(SYSTEM_LOG, EXT);

  public static final String MODULE_START = "geocoding module start";

  public static final String MODULE_READ_CFG = "reading config file (geoConfig.prop)";

  public static final String MODULE_CREATE_SOCKET = "creating server socket";

  public static final String MODULE_START_SERVICE = "starting service";

  public static final String MODULE_CONNECT_DB = "connecting to database";

  public static final String MODULE_DISCONNECT_DB = "diconnecting to database";

  public static final String MODULE_TERMINATED = "geocoding module terminated";

  public static final String ERROR_CFG_NO_EXIST = "config file does not exist";

  public static final String ERROR_CFG_BROKEN = "config file is broken";

  public static final String ERROR_CFG_CANT_READ = "cant read config file";

  public static final String ERROR_SOCKET = "server socket error";

  public static final String ERROR_DB_CONNECT_FAIL = "cannot connect to database";

  public static final String ERROR_DB_DISCONNECT_FAIL = "cannot disconnect to database";

  public static final String ERROR_STREAM_CANT_READ = "cannot read from input stream";

  public static final String ERROR_STREAM_CANT_WRITE = "cannnot write to output stream";

  public static final String WARNING_API_CONNECT_FAIL = "cannot connect to api";

  public static final String WARNING_DB_INSERT_FAIL = "cannot insert data";

}
```

```java
package setting;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.util.Properties;

import log.SystemLogger;

/**
 * Geocoding Moduleの設定ファイルの情報を格納するオブジェクト
 *
 * @author yuki
 * @version 1.0
 */
public class GeocodeSettings {

  private final String PROPERTY_FILE_NAME = "geoConfig.prop";

  private int modulePort; // GeocodingModuleが使用するポート番号

  private String requestEncode; // 入出力用のエンコード

  private int requestWaitTime; // 外部の緯度経度算出サービス問い合わせ時の待ち時間

  private int returnMax; // 返す緯度経度情報ペアの数

  private int cacheExpiration; // キャッシュテーブルの更新期間

  private int searchKeyExpiration; // リクエストテーブルの更新期間

  private String apiURL; // 外部の緯度経度算出サービスURL

  private String apiEncode; // 外部の緯度経度算出サービス問い合わせ時のエンコード

  private String logEncode; // ログの出力エンコード

  private String logLevel; // システムログの出力設定

  private String systemLogPath; // システムログの出力パス

  private String accessLogPath; // リクエストログの出力パス

  private String DBURL; // データベースのURL

  private String DBUserName; // データベースのユーザ名

  private String DBPass; // データベースのパスワード

  /**
   * 設定ファイルを読み込む
   *
   * @return 設定ファイルの読み込みに成功すればtrueを、失敗すればfalseを返す
   */
  public boolean readConfigFile() {
    FileInputStream input = null;
    try {
      String configFileAddress = initPath(PROPERTY_FILE_NAME); // コンフィグファイルのあるパスを取得する
      input = new FileInputStream(configFileAddress);
      Properties prop = new Properties();
      prop.load(input);

      modulePort = Integer.parseInt(prop.getProperty("modulePort"));
```

```
67        requestEncode = prop.getProperty("requestEncode");
68        requestWaitTime = Integer.parseInt(prop
69            .getProperty("requestWaitTime"));
70        returnMax = Integer.parseInt(prop.getProperty("returnMax"));
71        cacheExpiration = Integer.parseInt(prop
72            .getProperty("cacheExpiration"));
73        searchKeyExpiration = Integer.parseInt(prop
74            .getProperty("searchKeyExpiration"));
75        apiURL = prop.getProperty("apiURL");
76        apiEncode = prop.getProperty("apiEncode");
77
78        logEncode = prop.getProperty("logEncode");
79        logLevel = prop.getProperty("logLevel");
80        systemLogPath = initPath(prop.getProperty("systemLogPath"));
81        accessLogPath = initPath(prop.getProperty("accessLogPath"));
82
83        DBURL = prop.getProperty("DBURL");
84        DBUserName = prop.getProperty("DBUserName");
85        DBPass = prop.getProperty("DBPass");
86
87        input.close();
88        return true;
89      }
90      // 例外処理
91      catch (Exception e) {
92        SystemLogger.writer.writeError(SystemLogger.ERROR_CFG_BROKEN, e);
93        // inputを開いている途中の場合は、閉じる
94        if (input != null) {
95          try {
96            input.close();
97          } catch (IOException e1) {
98          }
99        }
100       return false;
101     }
102   }
103
104   /**
105    * 設定ファイルが壊れてないかチェックする
106    *
107    * @return 設定ファイルが正常な場合はtrueを、壊れている場合はfalseを返す
108    */
109   public boolean checkConfigFile() {
110     return notNull() && checkPortNumber(modulePort)
111         && isNaturalNumber(cacheExpiration)
112         && isNaturalNumber(searchKeyExpiration)
113         && isNaturalNumber(returnMax)
114         && isNaturalNumber(requestWaitTime) && logLevelCheck(logLevel)
115         && encodeCheck(logEncode) && encodeCheck(apiEncode)
116         && encodeCheck(requestEncode) && createDir(systemLogPath)
117         && createDir(accessLogPath);
118   }
119
120   /**
121    * 変数に値が入っているかどうかをチェックする
122    *
123    * @return
124    */
125   private boolean notNull() {
126     return apiURL != null && logLevel != null && systemLogPath != null
127         && logEncode != null && accessLogPath != null
128         && apiEncode != null && requestEncode != null && DBURL != null
129         && DBUserName != null && DBPass != null;
130   }
131
132   /**
133    * ポート番号が有効なものであるかどうかチェックする
```

```
134     *
135     * @param port
136     *            ポート番号
137     * @return ポート番号が有効範囲内ならtrueを、範囲外だったらfalseを返す
138     */
139    private boolean checkPortNumber(int port) {
140      return port > 0 && port <= 65535;
141    }
142
143    /**
144     * 数値が自然数であるかどうかチェックする
145     *
146     * @param num
147     *            数値
148     * @return 数値が自然数ならtrueを、そうでないならばfalseを返す
149     */
150    private boolean isNaturalNumber(int num) {
151      return num > 0;
152    }
153
154    /**
155     * 文字列の内容がonかoffかどうかチェックする
156     *
157     * @param input
158     *            文字列
159     * @return 文字列の内容が"on"か"off"である場合trueを、そうでない場合falseを返す
160     */
161    private boolean logLevelCheck(String input) {
162      return input.equals("DEBUG") || input.equals("INFO")
163          || input.equals("WARNING") || input.equals("ERROR");
164    }
165
166    /**
167     * パス名からディレクトリを作成する
168     *
169     * @param path
170     *            パス名
171     * @return ディレクトリの作成に成功した場合trueを、失敗した場合はfalseを返す。
172     *         指定したディレクトリが既に存在する場合は、trueを返す。
173     */
174    public boolean createDir(String path) {
175      try {
176        if (path == null)
177          return false;
178        if (pathExist(path) && dirCheck(path))
179          return true;
180        return (new File(path)).mkdir();
181      } catch (Exception e) {
182        return false;
183      }
184    }
185
186    /**
187     * パスが存在するかどうかチェックする
188     *
189     * @param path
190     *            パス名
191     * @return 指定したパスが存在する場合trueを、ない場合falseを返す
192     */
193    private boolean pathExist(String path) {
194      return (new File(path)).exists();
195    }
196
197    /**
198     * パスがディレクトリかどうかチェックする
199     *
200     * @param path
```

```
201     *            パス名
202     * @return ディレクトリである場合trueを、そうでない場合はfalseを返す
203     */
204    private boolean dirCheck(String path) {
205      return (new File(path)).isDirectory();
206    }
207
208    /**
209     * 文字コードが正当なものであるかチェックする
210     *
211     * @param encode
212     *            文字エンコード
213     * @return 正当な文字コードの場合trueを、不正な場合はfalseを返す
214     */
215    private boolean encodeCheck(String encode) {
216      String filename = "./GeoTest.temp";
217      File tempFile = new File(filename);
218      try {
219        FileOutputStream fOut = new FileOutputStream(filename);
220        OutputStreamWriter oStream = new OutputStreamWriter(fOut, encode);
221        BufferedWriter bWriter = new BufferedWriter(oStream);
222        String message = "This is a test";
223        bWriter.write(message);
224        bWriter.close();
225        oStream.close();
226        fOut.close();
227        return tempFile.delete();
228      } catch (Exception e) {
229        tempFile.delete();
230        return false;
231      }
232    }
233
234    /**
235     * パスを通す（どこからでも呼べるように、環境変数が設定されている場合は呼ぶ）
236     *
237     * @param path
238     * @return
239     */
240    private String initPath(String path) {
241      String g_home = System.getenv("G_HOME");// 環境変数からＧ＿ＨＯＭＥを取得する
242      if (g_home != null) {
243        path = g_home + "/" + path;// パスを通す
244      }
245      return path;
246    }
247
248    /**************************************************************************
249     * ゲッターメソッド郡
250     *************************************************************************/
251
252    public int getCacheExpiration() {
253      return cacheExpiration;
254    }
255
256    public int getModulePort() {
257      return modulePort;
258    }
259
260    public String getLogLevel() {
261      return logLevel;
262    }
263
264    public String getLogEncode() {
265      return logEncode;
266    }
267
```

```
268   public String getSystemLogPath() {
269     return systemLogPath;
270   }
271
272   public String getAccessLogPath() {
273     return accessLogPath;
274   }
275
276   public String getApiURL() {
277     return apiURL;
278   }
279
280   public int getReturnMax() {
281     return returnMax;
282   }
283
284   public String getDBURL() {
285     return DBURL;
286   }
287
288   public String getApiEncode() {
289     return apiEncode;
290   }
291
292   public String getRequestEncode() {
293     return requestEncode;
294   }
295
296   public int getSearchKeyExpiration() {
297     return searchKeyExpiration;
298   }
299
300   public int getRequestWaitTime() {
301     return requestWaitTime;
302   }
303
304   public String getDBUserName() {
305     return DBUserName;
306   }
307
308   public String getDBPass() {
309     return DBPass;
310   }
311 }
```

```java
package test;

import main.GeocodingModule;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;

/**
 * GeocodingModuleのテスト
 */
public class TestGeocodingModule {

  GeocodingModule module = new GeocodingModule();

  @BeforeClass
  public static void setUpBeforeClass() throws Exception {
  }

  @AfterClass
  public static void tearDownAfterClass() throws Exception {
  }

  @Before
  public void setUp() throws Exception {
  }

  @After
  public void tearDown() throws Exception {
  }
}
```

```
 1 package test;
 2
 3 import geocode.Geocoder;
 4
 5 import java.sql.Connection;
 6 import java.sql.DriverManager;
 7 import java.sql.ResultSet;
 8 import java.sql.Statement;
 9 import java.util.Date;
10
11 import junit.framework.TestCase;
12
13 import org.junit.After;
14 import org.junit.Before;
15 import org.junit.BeforeClass;
16
17 import setting.GeocodeSettings;
18
19 /**
20   * Geocoderのテスト
21   */
22 public class TestGeocoder extends TestCase {
23
24   static Geocoder geocoder;
25
26   static Statement stat;
27
28   //リクエストを受け、それに対応する文字列が作成できたかのテスト
29   public void testGeocode() {
30     // 検索キー関連のテスト
31     {
32       // 検索キーに対応する検索がされていなかった場合、検索キーが追加されている
33       geocoder.geocode("横浜");
34       String result = getValueFromDB("geocodingrequest", "searchquery",
35           "searchquery", "横浜");
36       assertEquals("横浜", result);
37       // 検索キーに対応する検索がされており、更新期間が過ぎている場合、検索キーを更新する
38       geocoder.geocode("江戸");
39       Date date = getDateFromDB("GeocodingRequest", "lastUpdatedRequest",
40           "searchquery", "江戸");
41       assertTrue(new Date().after(date));
42       // 検索キーに対応する検索がされており、更新期間が過ぎていない場合、検索キー回数以外更新
しない
43       date = getDateFromDB("GeocodingRequest", "lastUpdatedRequest",
44           "searchquery", "東京");
45       geocoder.geocode("東京");
46       Date newDate = getDateFromDB("GeocodingRequest",
47           "lastUpdatedRequest", "searchquery", "東京");
48       assertEquals(date, newDate);
49     }
50
51     setTestDatas();// 一度リセットする
52
53     // 地理情報関連のテスト
54     {
55       // 検索キーに対応する地理情報が存在しなかった場合、地理情報が追加されている
56       geocoder.geocode("横浜");
57       String result = getValueFromDB("GeocodingData", "locationname",
58           "locationname", "横浜");
59       assertEquals("横浜", result);
60       // 検索キーに対応する地理情報が存在し、更新期間が過ぎている場合、地理情報が更新されてい
る
61       geocoder.geocode("江戸");
62       Date date = getDateFromDB("GeocodingData", "lastUpdated",
63           "locationname", "江戸");
64       assertTrue(new Date().after(date));
65       // 検索キーに対応する地理情報が存在し、更新期間が過ぎていない場合、変更をしない
```

```
66        date = getDateFromDB("GeocodingData", "lastUpdated",
67            "locationname", "東京");
68        geocoder.geocode("東京");
69        Date newDate = getDateFromDB("GeocodingData", "lastUpdated",
70            "locationname", "東京");
71        assertEquals(date, newDate);
72      }
73
74      setTestDatas();// 一度リセットする
75
76      // 検索結果関連
77      {
78        // 検索キーに対応する地理情報が単数の場合、x,yの形で返す
79        assertEquals("100,200", geocoder.geocode("東京"));
80        // 検索キーに対応する地理情報が複数の場合、x,y:x,yの形で返す
81        assertEquals("200,300;300,400", geocoder.geocode("三田"));
82        // 検索キーに対応する地理情報が無い場合、nullを返す
83        assertEquals(null, geocoder.geocode("あああああああああ"));
84      }
85    }
86
87    /*************************************************************************
88     *
89     * テスト用意用メソッド郡
90     *
91     ************************************************************************/
92
93    @BeforeClass
94    public void createStatement() throws Exception {
95      // インスタンスを生成する
96      GeocodeSettings settings = new GeocodeSettings();
97      settings.readConfigFile();
98      // DBの設定をする（直打ち）
99      geocoder = new Geocoder(settings);
100     Class.forName("org.gjt.mm.mysql.Driver");
101     Connection c = DriverManager.getConnection(settings.getDBURL(),
102         settings.getDBUserName(), settings.getDBPass());
103     stat = c.createStatement();
104   }
105
106   @Before
107   public void setUp() throws Exception {
108     createStatement();
109     setTestDatas();
110   }
111
112   @After
113   public void tearDown() throws Exception {
114     stat.close();
115   }
116
117   /**
118    * DBを初期化する
119    */
120   private void setTestDatas() {
121     try {
122       String searchQuery;
123       String locationName;
124
125       // DBを初期化する
126       String sql = "delete from geocodingRequest";
127       stat.executeUpdate(sql);
128       sql = "delete from geocodingLink";
129       stat.executeUpdate(sql);
130       sql = "delete from geocodingData";
131       stat.executeUpdate(sql);
132
```

```
133         // データを用意する
134         // 検索キー関連
135         {
136           // 期限が切れていない場合
137           {
138             // 単独の場合
139             searchQuery = "東京";
140             locationName = "東京";
141             java.sql.Date date = new java.sql.Date(new Date().getTime()
142                 - 1000L * 24L * 60L * 60L);
143             sql = "insert into geocodingRequest values ('"
144                 + searchQuery + "',1,'" + date + "')";
145             stat.executeUpdate(sql);
146             sql = "insert into geocodingLink values ('" + searchQuery
147                 + "','" + locationName + "',1)";
148             stat.executeUpdate(sql);
149             // 複数存在する場合
150             searchQuery = "三田";
151             locationName = "三田駅 (東京) ";
152             date = new java.sql.Date(new Date().getTime());
153             sql = "insert into geocodingRequest values ('"
154                 + searchQuery + "',2,'" + date + "')";
155             stat.executeUpdate(sql);
156             sql = "insert into geocodingLink values ('" + searchQuery
157                 + "','" + locationName + "',1)";
158             stat.executeUpdate(sql);
159             locationName = "三田駅 (兵庫) ";
160             sql = "insert into geocodingLink values ('" + searchQuery
161                 + "','" + locationName + "',2)";
162             stat.executeUpdate(sql);
163           }
164           // 期限が切れている場合
165           searchQuery = "江戸";
166           locationName = "江戸";
167           Date date = new java.sql.Date(0);
168           sql = "insert into geocodingRequest values ('" + searchQuery
169               + "',1,'" + date + "')";
170           stat.executeUpdate(sql);
171           sql = "insert into geocodingLink values ('" + searchQuery
172               + "','" + locationName + "',1)";
173           stat.executeUpdate(sql);
174
175         }
176         // 地理情報関連
177         {
178           // 期限が切れていない場合
179           {
180             java.sql.Date date = new java.sql.Date(new Date().getTime());
181             // 単独の場合
182             locationName = "東京";
183             sql = "insert into geocodingData values ('" + locationName
184                 + "',100,200,'" + date + "','" + date + "')";
185             stat.executeUpdate(sql);
186             // 複数の場合
187             locationName = "三田駅 (東京) ";
188             sql = "insert into geocodingData values ('" + locationName
189                 + "',200,300,'" + date + "','" + date + "')";
190             stat.executeUpdate(sql);
191             locationName = "三田駅 (兵庫) ";
192             sql = "insert into geocodingData values ('" + locationName
193                 + "',300,400,'" + date + "','" + date + "')";
194             stat.executeUpdate(sql);
195           }
196           // 期限が切れている場合
197           Date date = new java.sql.Date(0);
198           locationName = "江戸";
199           sql = "insert into geocodingData values ('" + locationName
```

```
200             + "',100,200,'" + date + "','" + date + "')";
201         stat.executeUpdate(sql);
202       }
203     } catch (Exception e) {
204       e.printStackTrace();
205     }
206   }
207
208   /**
209    * DBから取得したい値をゲットする
210    */
211   private String getValueFromDB(String table, String target, String query,
212       String request) {
213     try {
214       // 取得する
215       String sql = "select " + target + " from " + table + " where "
216           + query + " ='" + request + "'";
217       ResultSet rs = stat.executeQuery(sql);
218
219       // 取得できたか調べる
220       while (rs.next()) {
221         return rs.getString(target);
222       }
223       return null;
224     } catch (Exception e) {
225       e.printStackTrace();
226       return null;
227     }
228   }
229
230   /**
231    * DBから取得したい日付をゲットする
232    */
233   private Date getDateFromDB(String table, String target, String query,
234       String request) {
235     try {
236       // 取得する
237       String sql = "select " + target + " from " + table + " where "
238           + query + " = '" + request + "'";
239       ResultSet rs = stat.executeQuery(sql);
240
241       // 取得できたか調べる
242       while (rs.next()) {
243         return rs.getDate(target);
244       }
245       return null;
246     } catch (Exception e) {
247       e.printStackTrace();
248       return null;
249     }
250   }
251 }
```

```
 1 package test;
 2
 3 import geocode.DBManager;
 4 import geocode.GeoData;
 5
 6 import java.sql.Connection;
 7 import java.sql.DriverManager;
 8 import java.sql.ResultSet;
 9 import java.sql.SQLException;
10 import java.sql.Statement;
11 import java.util.ArrayList;
12 import java.util.Date;
13 import java.util.List;
14
15 import org.junit.Before;
16 import junit.framework.TestCase;
17 import setting.GeocodeSettings;
18
19 /**
20  * データベースマネージャのテストプログラム
21  *
22  * @author Hideyuki Hirasawa
23  * @version 0.1
24  */
25 public class TestDBManager extends TestCase {
26
27   GeocodeSettings settings;
28
29   DBManager testManager;
30
31   final String DB_URL =
   "jdbc:mysql:///GeocodingModuleDB?useUnicode=true&characterEncoding=UTF8";
32
33   final String KEY_SING = "しんかんせん";
34
35   final String KEY_MULTI = "府中";
36
37   final String KEY_NONE = "冥王星";
38
39   final String DATA_SING = "京都府京都市下京区京都駅新幹線";
40
41   final String DATA_MULTIA = "府中駅（東京）";
42
43   final String DATA_MULTIB = "府中駅（徳島）";
44
45   final String DATA_MULTIC = "府中駅（広島）";
46
47   /**
48    * サーチキーがデータベースに存在するかどうか確かめる
49    */
50   public final void testKeyExist() {
51
52     try {
53       // 一時変数
54       boolean result;
55       String sql;
56       ResultSet rs;
57       int count;
58
59       // データベースに接続する
60       Class.forName("org.gjt.mm.mysql.Driver");
61       Connection c = DriverManager.getConnection(DB_URL, "root", "");
62       Statement stat = c.createStatement();
63
64       // キーがデータベースに存在する
65       addRequestData(KEY_SING, (new Date().getTime()));
66       result = testManager.keyExist(KEY_SING);
```

```
67        sql = "SELECT * FROM geocodingRequest WHERE searchQuery = '"
68            + KEY_SING + "'";
69        rs = stat.executeQuery(sql);
70        count = sizeResultSet(rs);
71        if (result == false || count != 1)
72          fail();
73
74        // キーがデータベース存在しない
75        deleteTempData();
76        result = testManager.keyExist(KEY_SING);
77        rs = stat.executeQuery(sql);
78        count = sizeResultSet(rs);
79        if (result == true || count != 0)
80          fail();
81
82      } catch (SQLException e) {
83        e.printStackTrace();
84      } catch (ClassNotFoundException e) {
85        e.printStackTrace();
86      }
87
88    }
89
90    /**
91     * 検索キーの検索回数を更新する
92     */
93    public final void testUpdateRequestTimes() {
94      int times = testManager.getRequestTimes(KEY_SING);
95      testManager.updateRequestTimes(KEY_SING);
96      addRequestData(KEY_SING, 0);
97      assertEquals(times + 1, testManager.getRequestTimes(KEY_SING));
98    }
99
100   /**
101    * サーチキーの有効期限を確認する
102    */
103   public final void testKeyExpired() {
104
105     // 期限が切れている場合
106     addRequestData(KEY_SING, 0);
107     assertTrue(testManager.keyExpired(KEY_SING));
108
109     // 期限が切れてない場合
110     deleteTempData();
111     addRequestData(KEY_SING, (new Date().getTime()));
112     assertFalse(testManager.keyExpired(KEY_SING));
113
114   }
115
116   /**
117    * サーチキーに該当する地名リストをデータベースに追加できるかどうか確かめる
118    */
119   public final void testAddKey() {
120
121     try {
122
123       // 一時変数
124       List<String> keyList = new ArrayList<String>();
125       boolean result;
126       String sql;
127       ResultSet rs;
128       int count;
129
130       // データベースに接続する
131       Class.forName("org.gjt.mm.mysql.Driver");
132       Connection c = DriverManager.getConnection(DB_URL, "root", "");
133       Statement stat = c.createStatement();
```

```
134
135       // 地名リストの長さが1の場合
136       keyList.add(DATA_SING);
137       testManager.addKey(KEY_SING, keyList);
138       result = testManager.keyExist(KEY_SING);
139       sql = "SELECT * FROM geocodingLink WHERE searchQuery = '"
140           + KEY_SING + "'";
141       rs = stat.executeQuery(sql);
142       count = sizeResultSet(rs);
143       if (result == false || count != 1)
144         fail();
145
146       // 地名リストの長さがNの場合
147       keyList = new ArrayList<String>();
148       keyList.add(DATA_MULTIA);
149       keyList.add(DATA_MULTIB);
150       keyList.add(DATA_MULTIC);
151       testManager.addKey(KEY_MULTI, keyList);
152       result = testManager.keyExist(KEY_MULTI);
153       sql = "SELECT * FROM geocodingLink WHERE searchQuery = '"
154           + KEY_MULTI + "'";
155       rs = stat.executeQuery(sql);
156       count = sizeResultSet(rs);
157       if (result == false || count != 3)
158         fail();
159
160       // 地名リストがない場合
161       testManager.addKey(KEY_NONE, null);
162       result = testManager.keyExist(KEY_NONE);
163       sql = "SELECT * FROM geocodingLink WHERE searchQuery = '"
164           + KEY_NONE + "'";
165       rs = stat.executeQuery(sql);
166       count = sizeResultSet(rs);
167       if (result == false || count != 0)
168         fail();
169
170     } catch (SQLException e) {
171       e.printStackTrace();
172     } catch (ClassNotFoundException e) {
173       e.printStackTrace();
174     }
175
176   }
177
178   /**
179    * サーチキーに該当する地名リストを更新できるかどうか確かめる
180    */
181   public final void testUpdateKey() {
182
183     try {
184
185       // 一時変数
186       List<String> keyList = new ArrayList<String>();
187       boolean result;
188       String sql;
189       ResultSet rs;
190       int count;
191
192       // 前処理
193       Class.forName("org.gjt.mm.mysql.Driver");
194       Connection c = DriverManager.getConnection(DB_URL, "root", "");
195       Statement stat = c.createStatement();
196       addRequestData(KEY_SING, (new Date(0).getTime()));
197       addRequestData(KEY_MULTI, (new Date(0).getTime()));
198       addRequestData(KEY_NONE, (new Date(0).getTime()));
199
200       // 地名リストの長さが1の場合
```

```
201       keyList.add(DATA_SING);
202       testManager.updateKey(KEY_SING, keyList);
203       result = testManager.keyExpired(KEY_SING);
204       sql = "SELECT * FROM geocodingLink WHERE searchQuery = '"
205           + KEY_SING + "'";
206       rs = stat.executeQuery(sql);
207       count = sizeResultSet(rs);
208       if (result == true || count != 1)
209         fail();
210
211       // 地名リストの長さがNの場合
212       keyList.clear();
213       keyList.add(DATA_MULTIA);
214       keyList.add(DATA_MULTIB);
215       keyList.add(DATA_MULTIC);
216       testManager.updateKey(KEY_MULTI, keyList);
217       result = testManager.keyExpired(KEY_MULTI);
218       sql = "SELECT * FROM geocodingLink WHERE searchQuery = '"
219           + KEY_MULTI + "'";
220       rs = stat.executeQuery(sql);
221       count = sizeResultSet(rs);
222       if (result == true || count != 3)
223         fail();
224
225       // 地名リストがない場合
226       testManager.updateKey(KEY_NONE, null);
227       result = testManager.keyExpired(KEY_NONE);
228       sql = "SELECT * FROM geocodingLink WHERE searchQuery = '"
229           + KEY_NONE + "'";
230       rs = stat.executeQuery(sql);
231       count = sizeResultSet(rs);
232       if (result == true || count != 0)
233         fail();
234
235     } catch (SQLException e) {
236       e.printStackTrace();
237     } catch (ClassNotFoundException e) {
238       e.printStackTrace();
239     }
240
241   }
242
243   /**
244    * 地名に該当する地理情報がデータベースに存在するかどうか確かめる
245    */
246   public final void testDataExist() {
247
248     try {
249
250       // 一時変数
251       boolean result;
252       String sql;
253       ResultSet rs;
254       int count;
255
256       // データベースに接続する
257       Class.forName("org.gjt.mm.mysql.Driver");
258       Connection c = DriverManager.getConnection(DB_URL, "root", "");
259       Statement stat = c.createStatement();
260
261       // 地理情報がデータベースに存在する
262       addGeocodeData(DATA_SING, (new Date().getTime()));
263       result = testManager.dataExist(DATA_SING);
264       sql = "SELECT * FROM geocodingData WHERE locationName = '"
265           + DATA_SING + "'";
266       rs = stat.executeQuery(sql);
267       count = sizeResultSet(rs);
```

```
268       if (result == false || count != 1)
269         fail();
270
271       // 地理情報がデータベース存在しない
272       deleteTempData();
273       result = testManager.dataExist(DATA_SING);
274       rs = stat.executeQuery(sql);
275       count = sizeResultSet(rs);
276       if (result == true || count != 0)
277         fail();
278
279     } catch (SQLException e) {
280       e.printStackTrace();
281     } catch (ClassNotFoundException e) {
282       e.printStackTrace();
283     }
284
285   }
286
287   /**
288    * 地理情報の有効期限を確かめる
289    */
290   public final void testDataExpired() {
291
292     // 期限が切れている場合
293     addGeocodeData(DATA_SING, 0);
294     assertTrue(testManager.dataExpired(DATA_SING));
295
296     // 期限が切れてない場合
297     deleteTempData();
298     addGeocodeData(DATA_SING, (new Date().getTime()));
299     assertFalse(testManager.dataExpired(DATA_SING));
300
301   }
302
303   /**
304    * 地理情報をデータベースに追加できるかどうか確かめる
305    */
306   public final void testAddData() {
307     GeoData data;
308     data = new GeoData(DATA_SING, "100", "100");
309     testManager.addData(data);
310     assertTrue(testManager.dataExist(DATA_SING));
311   }
312
313   /**
314    * 地理情報を更新できるかどうか確かめる
315    */
316   public final void testUpdateData() {
317     GeoData data;
318     addGeocodeData(DATA_SING, 0);
319     data = new GeoData(DATA_SING, "100", "100");
320     testManager.updateData(data);
321     assertFalse(testManager.dataExpired(DATA_SING));
322   }
323
324   /**
325    * サーチキーに該当する地名リストを取得できるかどうか確かめる
326    */
327   public final void testGetLocationNameList() {
328
329     // 一時変数
330     List<String> keyList = new ArrayList<String>();
331     List<String> list = new ArrayList<String>();
332
333     // 取得できる地名リストの長さが1の場合
334     keyList.add(DATA_SING);
```

```
335     testManager.addKey(KEY_SING, keyList);
336     list = testManager.getLocationNameList(KEY_SING);
337     assertEquals(1, list.size());
338 
339     // 取得できる地名リストの長さがNの場合
340     keyList = new ArrayList<String>();
341     keyList.add(DATA_MULTIA);
342     keyList.add(DATA_MULTIB);
343     keyList.add(DATA_MULTIC);
344     testManager.addKey(KEY_MULTI, keyList);
345     list = testManager.getLocationNameList(KEY_MULTI);
346     assertEquals(settings.getReturnMax(), list.size());
347 
348     // 取得できる地理リストがない場合
349     testManager.addKey(KEY_NONE, null);
350     list = testManager.getLocationNameList(KEY_NONE);
351     assertEquals(0, list.size());
352 
353   }
354 
355   /**
356    * サーチキーに該当する地理情報を取得できるかどうか確かめる
357    */
358   public final void testGetGeodata() {
359 
360     // 一時変数
361     List<String> keyList = new ArrayList<String>();
362     GeoData data;
363     List<GeoData> list = new ArrayList<GeoData>();
364 
365     // 取得できる地理情報が1の場合
366     keyList.add(DATA_SING);
367     testManager.addKey(KEY_SING, keyList);
368     data = new GeoData(DATA_SING, "100", "100");
369     testManager.addData(data);
370     list = testManager.getGeodata(KEY_SING);
371     assertEquals(1, list.size());
372 
373     // 取得できる地理情報がNの場合
374     keyList.clear();
375     keyList.add(DATA_MULTIA);
376     keyList.add(DATA_MULTIB);
377     keyList.add(DATA_MULTIC);
378     testManager.addKey(KEY_MULTI, keyList);
379     data = new GeoData(DATA_MULTIA, "100", "100");
380     testManager.addData(data);
381     data = new GeoData(DATA_MULTIB, "100", "100");
382     testManager.addData(data);
383     data = new GeoData(DATA_MULTIC, "100", "100");
384     testManager.addData(data);
385     list = testManager.getGeodata(KEY_MULTI);
386     assertEquals(settings.getReturnMax(), list.size());
387 
388     // 取得できる地理情報がない場合
389     testManager.addKey(KEY_NONE, null);
390     list = testManager.getGeodata(KEY_NONE);
391     assertEquals(0, list.size());
392 
393   }
394 
395   /*************************************************************************
396    * テスト用意メソッド郡
397    ************************************************************************/
398 
399   @Before
400   public void setUp() throws Exception {
401     settings = new GeocodeSettings();
```

```
402     settings.readConfigFile();
403     testManager = new DBManager(settings);
404     deleteTempData();
405   }
406
407   /**
408    * データを削除する
409    */
410   void deleteTempData() {
411     try {
412
413       // データベースに接続する
414       Class.forName("org.gjt.mm.mysql.Driver");
415       Connection c = DriverManager.getConnection(DB_URL, "root", "");
416       Statement stat = c.createStatement();
417
418       // geocodingRequestからテストデータを削除する
419       String sql = "DELETE FROM geocodingRequest WHERE searchQuery = '"
420           + KEY_SING + "'";
421       stat.executeUpdate(sql);
422       sql = "DELETE FROM geocodingRequest WHERE searchQuery = '"
423           + KEY_MULTI + "'";
424       stat.executeUpdate(sql);
425       sql = "DELETE FROM geocodingRequest WHERE searchQuery = '"
426           + KEY_NONE + "'";
427       stat.executeUpdate(sql);
428
429       // geocodingLinkからテストデータを削除する
430       sql = "DELETE FROM geocodingLink WHERE searchQuery = '" + KEY_SING
431           + "'";
432       stat.executeUpdate(sql);
433       sql = "DELETE FROM geocodingLink WHERE searchQuery = '" + KEY_MULTI
434           + "'";
435       stat.executeUpdate(sql);
436       sql = "DELETE FROM geocodingLink WHERE searchQuery = '" + KEY_NONE
437           + "'";
438       stat.executeUpdate(sql);
439
440       // geocodingDataからテストデータを削除する
441       sql = "DELETE FROM geocodingData WHERE locationName = '"
442           + DATA_SING + "'";
443       stat.executeUpdate(sql);
444       sql = "DELETE FROM geocodingData WHERE locationName = '"
445           + DATA_MULTIA + "'";
446       stat.executeUpdate(sql);
447       sql = "DELETE FROM geocodingData WHERE locationName = '"
448           + DATA_MULTIB + "'";
449       stat.executeUpdate(sql);
450       sql = "DELETE FROM geocodingData WHERE locationName = '"
451           + DATA_MULTIC + "'";
452       stat.executeUpdate(sql);
453
454     } catch (SQLException e) {
455       e.printStackTrace();
456     } catch (ClassNotFoundException e) {
457       e.printStackTrace();
458     }
459   }
460
461   /**
462    * 検索キーを追加する
463    */
464   void addRequestData(String searchQuery, long time) {
465     try {
466       Class.forName("org.gjt.mm.mysql.Driver");
467       Connection c = DriverManager.getConnection(DB_URL, "root", "");
468       Statement stat = c.createStatement();
```

```
469       java.sql.Date date = new java.sql.Date(time);
470       String sql = "insert into geocodingRequest values ('" + searchQuery
471           + "',1,'" + date + "')";
472       stat.executeUpdate(sql);
473     } catch (SQLException e) {
474       e.printStackTrace();
475     } catch (ClassNotFoundException e) {
476       e.printStackTrace();
477     }
478   }
479
480   /**
481    * 地理情報を追加する
482    */
483   void addGeocodeData(String locationName, long time) {
484     try {
485       Class.forName("org.gjt.mm.mysql.Driver");
486       Connection c = DriverManager.getConnection(DB_URL, "root", "");
487       Statement stat = c.createStatement();
488       java.sql.Date date = new java.sql.Date(time);
489       String sql = "insert into geocodingData values ('" + locationName
490           + "',100,200,'" + date + "','" + date + "')";
491       stat.executeUpdate(sql);
492     } catch (SQLException e) {
493       e.printStackTrace();
494     } catch (ClassNotFoundException e) {
495       e.printStackTrace();
496     }
497   }
498
499   /**
500    * 該当件数を返す
501    */
502   int sizeResultSet(ResultSet rs) {
503     int count = 0;
504     try {
505       while (rs.next())
506         count++;
507     } catch (SQLException e) {
508       e.printStackTrace();
509     }
510     return count;
511   }
512 }
```

```java
package test;

import geocode.APIManager;
import geocode.GeoData;

import java.util.List;

import junit.framework.TestCase;

import org.junit.Before;

import setting.GeocodeSettings;

public class TestAPIManager extends TestCase {

  private static APIManager api = null;

  @Before
  public void setUp() {
    // インスタンスを生成する
    GeocodeSettings settings = new GeocodeSettings();
    settings.readConfigFile();
    // DBの設定をする（直打ち）
    api = new APIManager(settings);
  }

  /**
   * 検索キーから地名リストを取得できるかのテスト
   */
  public void testSearchLocation() {
    // 地名が一つしかない場合、その地名を取得できるか
    List<String> locationNames = api.getSearchList("東京");
    assertEquals(1, locationNames.size());
    assertEquals("東京", locationNames.get(0));
    // 地名が複数ある場合、その地名リストを取得できるか
    locationNames = api.getSearchList("三田");
    assertEquals(2, locationNames.size());
    assertEquals("三田駅（東京）", locationNames.get(0));
    assertEquals("三田駅（兵庫）", locationNames.get(1));
    // 地名が一つもない場合、地名を取得しないか
    assertNull(api.getSearchList("あああああああああ"));
  }

  /**
   * 地名から地理情報を取得できるか
   */
  public void testLocation() {
    // 地名がある場合、地理情報を返すか
    GeoData geoData = api.getLocation("三田（東京）");
    assertEquals("三田（東京）",geoData.getLocationName());
    assertNotNull(geoData.getLatitude());
    assertNotNull(geoData.getLongitude());
    // 地名がなかった場合、地理情報は空か
    geoData = api.getLocation("qwerty");
    assertEquals("qwerty",geoData.getLocationName());
    assertNull(geoData.getLatitude());
    assertNull(geoData.getLongitude());
  }
}
```

```java
package test;

import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;

import org.junit.After;
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;

import setting.GeocodeSettings;

public class TestGeocodeSettings {

  GeocodeSettings settings = new GeocodeSettings();

  @Test
  /**
   * コンフィグファイルから読み込めたかどうかを判定する機能のテスト
   */
  public void testReadConfigFile() {
    // コンフィグファイルが存在する場合、Trueを返す
    createConfigFileNormal();
    assertTrue(settings.readConfigFile());
    // コンフィグファイルが存在しない場合、Falseを返す
    settings = new GeocodeSettings();
    deleteFile();
    assertFalse(settings.readConfigFile());
    // コンフィグファイルの項目が欠けている場合、Falseを返す
    settings = new GeocodeSettings();
    createConfigFileLacking();
    assertFalse(settings.readConfigFile());
  }

  @Test
  /**
   * 期待通りの値を読み込むことができたかどうかを判定する機能のテスト
   */
  public void testCheckConfigFile() {
    // コンフィグが正常の場合の場合、Trueを返す
    createConfigFileNormal();
    assertTrue(settings.readConfigFile());
    assertTrue(settings.checkConfigFile());
    // コンフィグファイルが正常でない場合、Falseを返す
    settings = new GeocodeSettings();
    createConfigFileAbnormal();
    assertTrue(settings.readConfigFile());
    assertFalse(settings.checkConfigFile());
  }

  /**
   * 最初にバックアップをとっておく
   *
   * @throws Exception
   */
  @BeforeClass
  public static void setUpBeforeClass() throws Exception {
    java.io.File prop = new java.io.File("geoConfig.prop");
    java.io.File temp = new java.io.File("temp.prop");
    temp.delete();
    prop.renameTo(temp);
  }
```

```
68
69   /**
70    * 最後にバックアップを復元する
71    *
72    * @throws Exception
73    */
74   @AfterClass
75   public static void tearDownAfterClass() throws Exception {
76     java.io.File prop = new java.io.File("temp.prop");
77     java.io.File temp = new java.io.File("geoConfig.prop");
78     temp.delete();
79     prop.renameTo(temp);
80   }
81
82   @Before
83   public void setUp() throws Exception {
84   }
85
86   @After
87   public void tearDown() throws Exception {
88   }
89
90   /**
91    * 全ての項目が埋まったコンフィグファイルを出力する
92    */
93   private void createConfigFileNormal() {
94     try {
95       File prop = new File("geoConfig.prop");
96       prop.delete();
97       BufferedWriter logBuffer = new BufferedWriter(new FileWriter(prop,
98           true));
99       logBuffer
100          .write("modulePort=8080¥n"
101              + "requestEncode=UTF-8¥n"
102              + "requestWaitTime=5000¥n"
103              + "returnMax=2¥n"
104              + "cacheExpiration=15¥n"
105              + "searchKeyExpiration=15¥n"
106              + "apiURL=http://www.geocoding.jp/api/?q=¥n"
107              + "apiEncode=UTF-8¥n"
108              + "logEncode=UTF-8¥n"
109              + "logLevel=DEBUG¥n"
110              + "systemLogPath=./systemLog/¥n"
111              + "accessLogPath=./accessLog/¥n"
112              +
   "DBURL=jdbc:mysql:///geocodingmoduledb?useUnicode=true&characterEncoding=UTF8¥n"
113              + "DBUserName=root¥n" + "DBPass=¥n");
114       logBuffer.newLine();
115       logBuffer.close();
116     } catch (Exception e) {
117
118     }
119   }
120
121   /**
122    * ファイルを消す
123    */
124   private void deleteFile() {
125     File prop = new File("geoConfig.prop");
126     prop.delete();
127   }
128
129   /**
130    * 項目に対応するものが間違っているコンフィグファイルを出力する
131    */
132   private void createConfigFileAbnormal() {
133     try {
```

```
134       File prop = new File("geoConfig.prop");
135       prop.delete();
136       BufferedWriter logBuffer = new BufferedWriter(new FileWriter(prop,
137           true));
138       logBuffer
139           .write("modulePort=808000¥n"
140               + "requestEncode=UTF-8¥n"
141               + "requestWaitTime=5000¥n"
142               + "returnMax=2¥n"
143               + "cacheExpiration=15¥n"
144               + "searchKeyExpiration=15¥n"
145               + "apiURL=http://www.geocoding.jp/api/?q=¥n"
146               + "apiEncode=UTF-8¥n"
147               + "logEncode=UTF-8¥n"
148               + "logLevel=DEBUG¥n"
149               + "systemLogPath=./systemLog/¥n"
150               + "accessLogPath=./accessLog/¥n"
151               +
    "DBURL=jdbc:mysql:///geocodingmoduledb?useUnicode=true&characterEncoding=UTF8¥n"
152               + "DBUserName=root¥n" + "DBPass=¥n");
153       logBuffer.newLine();
154       logBuffer.close();
155     } catch (Exception e) {
156
157     }
158   }
159
160   /**
161    * 項目が欠けているコンフィグファイルを作成する ここではreturnMaxを抜かす
162    */
163   private void createConfigFileLacking() {
164     try {
165       File prop = new File("geoConfig.prop");
166       prop.delete();
167       BufferedWriter logBuffer = new BufferedWriter(new FileWriter(prop,
168           true));
169       logBuffer
170           .write("modulePort=8080¥n"
171               + "requestEncode=UTF-8¥n"
172               + "requestWaitTime=5000¥n"
173               + "cacheExpiration=15¥n"
174               + "searchKeyExpiration=15¥n"
175               + "apiURL=http://www.geocoding.jp/api/?q=¥n"
176               + "apiEncode=UTF-8¥n"
177               + "logEncode=UTF-8¥n"
178               + "logLevel=DEBUG¥n"
179               + "systemLogPath=./systemLog/¥n"
180               + "accessLogPath=./accessLog/¥n"
181               +
    "DBURL=jdbc:mysql:///geocodingmoduledb?useUnicode=true&characterEncoding=UTF8¥n"
182               + "DBUserName=root¥n" + "DBPass=¥n");
183       logBuffer.newLine();
184       logBuffer.close();
185     } catch (Exception e) {
186
187     }
188   }
189 }
```

```java
package test;

import static org.junit.Assert.assertTrue;

import java.io.File;

import log.LogWriter;
import log.SystemLogger;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class TestLogWriter {

  private String fileName;

  @Before
  public void setUp() throws Exception {
    fileName = SystemLogger.writer.getFilePath();
    SystemLogger.writer.setPath(".");
    LogWriter.setLogLevel(LogWriter.DEBUG);
  }

  @After
  public void tearDown() throws Exception {
    (new File(fileName)).delete();
  }

  /**
   * ログの出力・非出力をテストする
   */
  @Test
  public final void testEnable() {
    // ログの非出力をテストする
    LogWriter.setLogLevel(LogWriter.INFO);
    SystemLogger.writer.writeDebug("Test1");
    assertTrue(!(new File(fileName)).exists());

    // ログの出力をテストする
    SystemLogger.writer.writeError("Test2");
    assertTrue((new File(fileName)).exists());
  }

  /**
   * ログの出力機能をテストする
   */
  @Test
  public final void testWrite() {
    SystemLogger.writer.writeError("Test3", null);
    assertTrue((new File(fileName)).exists());
  }

  /**
   * ログの出力パス設定をテストする
   */
  @Test
  public final void testSetPath() {
    (new File("test_dir")).mkdir();
    SystemLogger.writer.setPath("test_dir");
    SystemLogger.writer.writeError("Test4", null);
    assertTrue((new File("test_dir" + File.separator + fileName)).exists());
    (new File("test_dir" + File.separator + fileName)).delete();
    (new File("test_dir")).delete();
  }


```

```
68 }
```

```
 1 package test;
 2
 3 import static org.junit.Assert.assertEquals;
 4 import static org.junit.Assert.assertTrue;
 5
 6 import geocode.DBManager;
 7 import geocode.GeoData;
 8 import geocode.GeoDataManager;
 9
10 import java.sql.Connection;
11 import java.sql.DriverManager;
12 import java.sql.ResultSet;
13 import java.sql.Statement;
14 import java.util.Date;
15 import java.util.List;
16
17 import org.junit.Before;
18 import org.junit.BeforeClass;
19 import org.junit.Test;
20
21 import setting.GeocodeSettings;
22
23 public class TestGeoDataManager {
24
25   private static GeoDataManager manager;
26
27   private static DBManager dbManager;
28
29   private static Statement stat;
30
31   /**
32    * 検索キーを更新するテスト
33    */
34   @Test
35   public void testUpdateDatabase() {
36     // 検索キー関連のテスト
37     {
38       // 検索キーに対応する検索がされていなかった場合、検索キーが追加されている
39       manager.updateDatabase("横浜", dbManager);
40       String result = getValueFromDB("GeocodingRequest", "searchQuery",
41           "searchQuery", "横浜");
42       assertEquals("横浜", result);
43       // 検索キーに対応する検索がされており、更新期間が過ぎている場合、検索キーを更新する
44       Date date = getDateFromDB("GeocodingRequest", "lastUpdatedRequest",
45           "searchQuery", "江戸");
46       manager.updateDatabase("江戸", dbManager);
47       Date newDate = getDateFromDB("GeocodingRequest",
48           "lastUpdatedRequest", "searchQuery", "江戸");
49       assertTrue(date.before(newDate));
50       // 検索キーに対応する検索がされており、更新期間が過ぎていない場合、検索キー回数以外更新
しない
51       date = getDateFromDB("GeocodingRequest", "lastUpdatedRequest",
52           "searchQuery", "東京");
53       manager.updateDatabase("東京", dbManager);
54       newDate = getDateFromDB("GeocodingRequest", "lastUpdatedRequest",
55           "searchQuery", "東京");
56       assertEquals(date, newDate);
57     }
58
59     setTestDatas();// 一度リセットする
60
61     // 地理情報関連のテスト
62     {
63       // 検索キーに対応する地理情報が存在しなかった場合、地理情報が追加されている
64       manager.updateDatabase("横浜", dbManager);
65       String result = getValueFromDB("GeocodingData", "locationName",
66           "locationName", "横浜");
```

```
67        assertEquals("横浜", result);
68        // 検索キーに対応する地理情報が存在し、更新期間が過ぎている場合、地理情報が更新されている
69        Date date = getDateFromDB("GeocodingData", "lastUpdated",
70            "locationName", "江戸");
71        manager.updateDatabase("江戸", dbManager);
72        Date newDate = getDateFromDB("GeocodingData", "lastUpdated",
73            "locationName", "江戸");
74        assertTrue(date.before(newDate));
75        // 検索キーに対応する地理情報が存在し、更新期間が過ぎていない場合、変更をしない
76        date = getDateFromDB("GeocodingData", "lastUpdated",
77            "locationName", "東京");
78        manager.updateDatabase("東京", dbManager);
79        newDate = getDateFromDB("GeocodingData", "lastUpdated",
80            "locationName", "東京");
81        assertEquals(date, newDate);
82      }
83    }
84
85    /**
86     * 地理情報を取得するテスト
87     */
88    @Test
89    public void TestGetGeoDatas() {
90      List<GeoData> geoDatas;
91      // 地理情報が1つ取得できる場合、1つリストに入った状態で取得できる
92      geoDatas = manager.getGeoDatas("東京", dbManager);
93      assertEquals(1, geoDatas.size());
94      assertEquals("東京", geoDatas.get(0).getLocationName());
95      // 地理情報が複数取得できる場合、複数リストに入った状態で取得できる
96      geoDatas = manager.getGeoDatas("三田", dbManager);
97      assertEquals(2, geoDatas.size());
98      assertEquals("三田駅（東京）", geoDatas.get(0).getLocationName());
99      assertEquals("三田駅（兵庫）", geoDatas.get(1).getLocationName());
100     // 地理情報が取得できない場合、空のリストが取得できる
101     geoDatas = manager.getGeoDatas("あああああ", dbManager);
102     assertEquals(0, geoDatas.size());
103   }
104
105   /*****************************************************************************
106    *
107    * テスト用意用メソッド郡
108    *
109    *****************************************************************************/
110
111   @BeforeClass
112   public static void setUpBeforeClass() throws Exception {
113     // インスタンスを生成する
114     GeocodeSettings settings = new GeocodeSettings();
115     settings.readConfigFile();
116     dbManager = new DBManager(settings);
117     // DBの設定をする（直打ち）
118     manager = new GeoDataManager(settings);
119     Class.forName("org.gjt.mm.mysql.Driver");
120     Connection c = DriverManager
121         .getConnection(
122             "jdbc:mysql:///GeocodingModuleDB?useUnicode=true&characterEncoding=UTF8",
123             "root", "");
124     stat = c.createStatement();
125   }
126
127   @Before
128   public void setUp() throws Exception {
129     setTestDatas();
130   }
131
132   /**
```

```
133     * DBを初期化する
134     */
135    private void setTestDatas() {
136      try {
137        String searchQuery;
138        String locationName;
139
140        // DBを初期化する
141        String sql = "delete from geocodingRequest";
142        stat.executeUpdate(sql);
143        sql = "delete from geocodingLink";
144        stat.executeUpdate(sql);
145        sql = "delete from geocodingData";
146        stat.executeUpdate(sql);
147
148        // データを用意する
149        // 検索キー関連
150        {
151          // 期限が切れていない場合
152          {
153            // 単独の場合
154            searchQuery = "東京";
155            locationName = "東京";
156            java.sql.Date date = new java.sql.Date(new Date().getTime()
157                - 1000L * 24L * 60L * 60L);
158            sql = "insert into geocodingRequest values ('"
159                + searchQuery + "',1,'" + date + "')";
160            stat.executeUpdate(sql);
161            sql = "insert into geocodingLink values ('" + searchQuery
162                + "','" + locationName + "',1)";
163            stat.executeUpdate(sql);
164            // 複数存在する場合
165            searchQuery = "三田";
166            locationName = "三田駅（東京）";
167            date = new java.sql.Date(new Date().getTime());
168            sql = "insert into geocodingRequest values ('"
169                + searchQuery + "',2,'" + date + "')";
170            stat.executeUpdate(sql);
171            sql = "insert into geocodingLink values ('" + searchQuery
172                + "','" + locationName + "',1)";
173            stat.executeUpdate(sql);
174            locationName = "三田駅（兵庫）";
175            sql = "insert into geocodingLink values ('" + searchQuery
176                + "','" + locationName + "',2)";
177            stat.executeUpdate(sql);
178          }
179          // 期限が切れている場合
180          searchQuery = "江戸";
181          locationName = "江戸";
182          Date date = new java.sql.Date(0);
183          sql = "insert into geocodingRequest values ('" + searchQuery
184              + "',1,'" + date + "')";
185          stat.executeUpdate(sql);
186          sql = "insert into geocodingLink values ('" + searchQuery
187              + "','" + locationName + "',1)";
188          stat.executeUpdate(sql);
189
190        }
191        // 地理情報関連
192        {
193          // 期限が切れていない場合
194          {
195            java.sql.Date date = new java.sql.Date(new Date().getTime());
196            // 単独の場合
197            locationName = "東京";
198            sql = "insert into geocodingData values ('" + locationName
199                + "',100,200,'" + date + "','" + date + "')";
```

```
200           stat.executeUpdate(sql);
201           // 複数の場合
202           locationName = "三田駅（東京）";
203           sql = "insert into geocodingData values ('" + locationName
204               + "',200,300,'" + date + "','" + date + "')";
205           stat.executeUpdate(sql);
206           locationName = "三田駅（兵庫）";
207           sql = "insert into geocodingData values ('" + locationName
208               + "',200,300,'" + date + "','" + date + "')";
209           stat.executeUpdate(sql);
210         }
211         // 期限が切れている場合
212         Date date = new java.sql.Date(0);
213         locationName = "江戸";
214         sql = "insert into geocodingData values ('" + locationName
215             + "',100,200,'" + date + "','" + date + "')";
216         stat.executeUpdate(sql);
217       }
218     } catch (Exception e) {
219       e.printStackTrace();
220     }
221   }
222
223   /**
224    * DBから取得したい値をゲットする
225    */
226   private String getValueFromDB(String table, String target, String rowName,
227       String request) {
228     try {
229       // 取得する
230       String sql = "select " + target + " from " + table + " where "
231           + rowName + " ='" + request + "'";
232       ResultSet rs = stat.executeQuery(sql);
233
234       // 取得できたか調べる
235       while (rs.next()) {
236         return rs.getString(target);
237       }
238       return null;
239     } catch (Exception e) {
240       e.printStackTrace();
241       return null;
242     }
243   }
244
245   /**
246    * DBから取得したい日付をゲットする
247    */
248   private Date getDateFromDB(String table, String target, String rowName,
249       String request) {
250     try {
251       // 取得する
252       String sql = "select " + target + " from " + table + " where "
253           + rowName + " ='" + request + "'";
254       ResultSet rs = stat.executeQuery(sql);
255
256       // 取得できたか調べる
257       while (rs.next()) {
258         return rs.getDate(target);
259       }
260       return null;
261     } catch (Exception e) {
262       e.printStackTrace();
263       return null;
264     }
265   }
266 }
```

```java
package test;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;
import java.net.UnknownHostException;

/**
 * ソケット通信のテスト用クライアント
 *
 * @author Toshiyuki Sato
 *
 */
public class TestClient {

  public static void main(String[] args) {
    TestClient testClient = new TestClient();
    testClient.doSocketConnection();
  }

  /**
   * ソケット通信を行う
   */
  public void doSocketConnection() {

    try {
      while (true) {


        // 接続する
        Socket socket = new Socket("localhost", 8000);// 本番環境用
        //Socket socket = new Socket("localhost", 18822);// テスト用

        // 入出力ストリームの作成
        // 文字入出力用インタフェースを宣言する
        PrintWriter output = new PrintWriter(new OutputStreamWriter(
            socket.getOutputStream(), "UTF-8"), true);
        BufferedReader input = new BufferedReader(
            new InputStreamReader(socket.getInputStream(), "UTF-8"));

        // サーバーへ数値を送信
        System.out.print("地名を入力して下さい：");
        String place = new BufferedReader(new InputStreamReader(
            System.in)).readLine();
        if (place.equals("exitexit")) {
          break;
        }
        output.println(place);

        // 結果を表示する
        String getString = input.readLine();
        if (getString != null && !getString.equals("") && !getString.equals("null")) {
          System.out.println("結果："+getString);
        } else {
          System.out.println("取得できませんでした");
        }
        System.out.println();

        // ストリームを閉じる
        input.close();
        output.close();
        // ソケットを閉じる
        socket.close();
      }
```

```
68      }
69
70      // 例外処理。コピペです
71      catch (UnknownHostException e) {
72        System.err.println("UnknownHost" + e);
73      } catch (IOException e) {
74        System.err.println("IOException: " + e);
75      }
76
77    }
78  }
```