LINE情報配信Bot 設計書 v2.0

毎朝の天気・ニュース・交通情報配信システム（拡張版）

📋 変更履歴

バージョン	日付	変更内容
v1.0	2026/01/12	初版リリース（基本機能）
v2.0	2026/01/12	天気アラート、ニュースカテゴリ、交通情報、花粉情報を追加

1. システム概要

Google Apps Script (GAS) を使用して、毎朝決まった時刻にLINE Messaging APIを通じて以下の情報を自動配信するシステムです。

v2.0 配信情報

- ✅ ユーザーのLINE登録名での挨拶
- ✅ 当日の日付と曜日
- ✅ 指定地域の天気予報と気温
- 🆕 **天気に応じたアラート（傘・防寒など）**
- 🆕 **選択可能なニュースカテゴリ（一般/テクノロジー/ビジネス/スポーツ/エンタメ）**
- 🆕 **指定路線の鉄道運行情報（遅延・運休アラート）**
- 🆕 **花粉情報（季節限定：2月〜5月）**

2. 機能定義

2.1 主要機能（v1.0からの継承）

F-01: スケジュール管理機能

- Googleスプレッドシートで配信のON/OFFを管理

- 地域コード、ニュースカテゴリ、路線名を設定
- 備考欄でスケジュールの目的を記録

F-02: 定期実行機能

- GASのトリガーを使用して毎朝決まった時刻に実行
- 複数のスケジュールを一括処理可能

F-03: ユーザー情報取得機能

- LINE Messaging APIからユーザーの表示名を取得
- パーソナライズされた挨拶文を生成

F-04: 天気情報取得機能

- 気象庁のAPIから天気予報を取得
- 当日の天気概況と最高気温・最低気温を取得
- 地域コードに基づいて地域別の天気を配信

F-05: ニュース取得機能

- GoogleニュースのRSSフィードから最新ニュースを取得
- トップ3件のニュースタイトルを抽出

F-06: メッセージ組み立て・送信機能

- 取得した情報を見やすいフォーマットで組み立て
- LINE Messaging APIでプッシュメッセージを送信

F-07: ログ記録機能

- 送信日時、スケジュールの備考、送信結果を記録
- トラブルシューティング用の履歴管理

2.2 新規機能（v2.0）

F-08: 天気アラート機能 🆕

- 降水確率が50%以上の場合、傘の持参を推奨
- 最高気温が30度以上の場合、熱中症注意
- 最低気温が5度以下の場合、防寒注意
- 天気概況に「雪」「大雨」「暴風」などのキーワードがある場合、警告メッセージ

F-09: ニュースカテゴリ選択機能 🆕

- 5つのカテゴリから選択可能
  - **一般**: 総合ニュース（デフォルト）
  - **テクノロジー**: IT・テクノロジー関連
  - **ビジネス**: 経済・ビジネス
  - **スポーツ**: スポーツ全般
  - **エンタメ**: エンターテイメント・芸能
- カテゴリ別のGoogleニュースRSSを使用

F-10: 鉄道運行情報取得機能 🆕

- Yahoo!路線情報のスクレイピングで運行情報を取得
- 遅延・運休・見合わせが発生している場合のみ表示
- 主要路線に対応（JR各線、私鉄各線）
- 正常運行時は「運行情報なし」を簡潔に表示

F-11: 花粉情報取得機能 🆕

- tenki.jpの花粉情報APIを使用
- 2月〜5月の花粉シーズンのみ自動表示
- 花粉飛散レベル（少ない/やや多い/多い/非常に多い）を表示
- 地域コードに基づいた地域別情報

---

3. 処理の流れ（アルゴリズム）

3.1 全体フロー（v2.0）

```
[開始：トリガー起動（例：毎朝7:00）]
  ↓
[スプレッドシート・ログシートを取得]
  ↓
[現在日時を取得し、日付・曜日・月を生成]
  ↓
[LINE設定情報を取得（トークン、ユーザーID）]
  ↓
```

```
[スケジュールシートの各行をループ処理] ———→ [次のスケジュール]
  ↓                                    ↑
[有効フラグがONか？] ― NO ――――――――――――――――――――┘
  ↓ YES
[LINEからユーザー名を取得]
  ↓
[天気情報を取得（地域コード使用）]
  ↓
[天気アラートを生成] 🆕
  ↓
[ニュースを取得（カテゴリ指定）] 🆕
  ↓
[鉄道運行情報を取得（路線名指定）] 🆕
  ↓
[花粉情報を取得（2-5月のみ）] 🆕
  ↓
[メッセージを組み立て]
  ↓
[LINEでメッセージ送信]
  ↓
[送信結果をログシートに記録]
  ↓
[全スケジュール処理完了？] ― NO ――――――――――┘
  ↓ YES
[終了]
```

3.2 詳細処理ステップ

ステップ1: 初期設定と準備

1. スプレッドシートを開く
2. 「スケジュール」シートと「ログ」シートを取得
3. 現在日時を取得
4. 日付文字列を「M月d日」形式で生成
5. 曜日を「〇曜日」形式で生成
6. **現在の月を取得（花粉情報の表示判定用）** 🆕

7. スクリプトプロパティからLINE_ACCESS_TOKENとUSER_IDを取得

ステップ2: スケジュールのループ処理（v2.0）

```
FOR 各行 IN スケジュールシート（2行目から）:
  [有効, 地域コード, ニュースカテゴリ, 路線名, 備考] = 行データ 🆕

  IF 有効 == FALSE:
    CONTINUE（次の行へ）

  // 以下、有効な場合のみ実行

  1. ユーザー名取得処理を実行
  2. 天気情報取得処理を実行（地域コード使用）
  3. 天気アラート生成処理を実行 🆕
  4. ニュース取得処理を実行（カテゴリ指定） 🆕
  5. 鉄道運行情報取得処理を実行（路線名指定） 🆕
  6. 花粉情報取得処理を実行（2-5月のみ） 🆕
  7. メッセージ組み立て処理を実行
  8. LINE送信処理を実行
  9. ログ記録処理を実行

END FOR
```

ステップ3: ユーザー名取得処理

1. LINE Profile APIのエンドポイントを構築
2. アクセストークンをヘッダーに設定してGETリクエスト
3. レスポンスJSONから `displayName` を抽出
4. エラー時は「ユーザー」という文字列を返す

ステップ4: 天気情報取得処理（拡張版）

1. 地域コードを6桁の文字列に変換（前ゼロ埋め）
2. 気象庁APIのエンドポイントにリクエスト
3. JSONレスポンスをパース
4. **以下の情報を構造化して取得:** 🆕

```
{  weather: "晴れ時々曇り",          // 天気概況
   maxTemp: 15,                    // 最高気温
   minTemp: 5,                     // 最低気温
   precipitation: 30,              // 降水確率（%）
      rawText: "晴れ時々曇り 所により雨" // 元のテキスト}
```

5. 降水確率の取得方法:
   - `json[0].timeSeries[1].areas[0].pops[0]` から取得
   - データが無い場合は0%とする
6. エラー時は null を返す

ステップ5: 天気アラート生成処理 🆕

```
function generateWeatherAlert(weatherData) {
  alerts = []

  // 降水確率チェック
  IF weatherData.precipitation >= 50:
    alerts.push("☔ 傘を忘れずに！（降水確率" + weatherData.precipitation + "%）")
  ELSE IF weatherData.precipitation >= 30:
    alerts.push("☁️ 傘があると安心です（降水確率" + weatherData.precipitation + "%）")

  // 気温チェック（高温）
  IF weatherData.maxTemp >= 30:
    alerts.push("🌡️ 熱中症に注意！こまめに水分補給を")
  ELSE IF weatherData.maxTemp >= 25:
    alerts.push("🌞 暑くなりそうです")

  // 気温チェック（低温）
  IF weatherData.minTemp <= 5:
    alerts.push("🧥 しっかり防寒してください（最低気温" + weatherData.minTemp + "度）")
  ELSE IF weatherData.minTemp <= 10:
    alerts.push("🍃 朝晩は冷えます。上着があると安心")
```

```
  // 天気概況の特殊キーワードチェック
  keywords = ["大雨", "暴風", "雪", "警報", "注意報", "雷"]
  FOR keyword IN keywords:
    IF weatherData.rawText.includes(keyword):
      alerts.push("⚠️ " + keyword + "に注意してください")
      BREAK

  IF alerts.length > 0:
    RETURN "\n⚡アラート⚡\n" + alerts.join("\n")
  ELSE:
    RETURN ""
}
```

アラート優先度:

1. 警報・注意報（最優先）
2. 降水確率50%以上
3. 極端な気温（30度以上 or 5度以下）
4. 注意レベルの天候（降水確率30%以上 or 気温25度以上 or 10度以下）

ステップ6: ニュース取得処理（カテゴリ対応版） 🆕

```
function getNews(category) {
  // カテゴリに応じたRSS URLを選択
  categoryUrls = {
    "一般": "https://news.google.com/rss?hl=ja&gl=JP&ceid=JP:ja",
    "テクノロジー": "https://news.google.com/rss/topics/CAAqJggKIiBDQkFTRWdvSUwyMHZNRGRqTVhZU0FtcGhHZ0pLVUNnQVAB?hl=ja&gl=JP&ceid=JP:ja",
    "ビジネス": "https://news.google.com/rss/topics/CAAqJggKIiBDQkFTRWdvSUwyMHZNRGx6TVdZU0FtcGhHZ0pLVUNnQVAB?hl=ja&gl=JP&ceid=JP:ja",
    "スポーツ": "https://news.google.com/rss/topics/CAAqJggKIiBDQkFTRWdvSUwyMHZNRFp1ZEdvU0FtcGhHZ0pLVUNnQVAB?hl=ja&gl=JP&ceid=JP:ja",
    "エンタメ": "https://news.google.com/rss/topics/CAAqJggKIiBDQkFTRWdvSUwyMHZNREpxYW5RU0FtcGhHZ0pLVUNnQVAB?hl=ja&gl=JP&ceid=JP:ja"
```

```
  }

  // カテゴリが未指定または無効な場合は「一般」を使用
  url = categoryUrls[category] || categoryUrls["一般"]

  TRY:
    response = UrlFetchApp.fetch(url)
    xml = response.getContentText()
    items = xml.split('<item>')
    newsList = []

    FOR i = 1 TO 3:
      IF items[i]:
        title = items[i].split('<title>')[1].split('</title>')[0]
        // 配信元情報を除去（" - ○○新聞"など）
        cleanTitle = title.split(' - ')[0]
        newsList.push(i + ". " + cleanTitle)

    RETURN newsList.join("\n")
  CATCH error:
    RETURN "ニュースの取得に失敗しました"
}
```

対応カテゴリとURL:

カテゴリ	説明	Google News トピックID
一般	総合ニュース	（トップページ）
テクノロジー	IT・テクノロジー	CAAqJggKIiBDQkFTRWdvSUwyMHZNRGRqTVhZ...
ビジネス	経済・ビジネス	CAAqJggKIiBDQkFTRWdvSUwyMHZNRGx6TVdZ...
スポーツ	スポーツ全般	CAAqJggKIiBDQkFTRWdvSUwyMHZNRFp1ZEdv...
エンタメ	芸能・エンタメ	CAAqJggKIiBDQkFTRWdvSUwyMHZNREpxYW5R...

ステップ7: 鉄道運行情報取得処理 🆕

```
function getTrainInfo(routeName) {
  if (!routeName || routeName.trim() === "") {
    return null;
```

```
}

// マスタシートからリストを読み込む処理
const ss = SpreadsheetApp.getActiveSpreadsheet();
const masterSheet = ss.getSheetByName("路線マスタ");
const masterData = masterSheet.getDataRange().getValues();

// シートのデータを変換
const routeMap = {};
for (let i = 1; i < masterData.length; i++) {
  const name = masterData[i][0]; // A列: 路線名
  const code = masterData[i][1]; // B列: コード
  routeMap[name] = code.toString();
}

const routeCode = routeMap[routeName];
if (!routeCode) {
  Logger.log("路線マスタに未登録: " + routeName);
  return {
    status: "未対応",
    detail: "この路線はまだ対応していません"
  };
}

try {
  const url = "https://transit.yahoo.co.jp/diainfo/" + routeCode + "/0";
  const response = UrlFetchApp.fetch(url);
  const html = response.getContentText();

  // 1. 平常運転の場合
  if (html.includes("icnNormalLarge") || html.includes("平常運転")) {
    return {
      status: "平常運転",
      detail: "現在、事故・遅延に関する情報はありません。"
    };
  }

  // 2. 異常がある場合、その理由（テキスト）を抜き出す
```

```
    // <dd class="trouble"> または <dd class="normal"> の中の <p>タグの中身を取得
    const statusMatch = html.match(/<dd class="(?:trouble|normal)">\s*<p>(.*?)<\/p>/);

    if (statusMatch) {
      const statusText = statusMatch[1].replace(/<[^>]+>/g, "").trim();
      return {
        status: "⚠️ 運行情報あり",
        detail: statusText
      };
    }

    return null;

  } catch (e) {
    Logger.log("解析エラー: " + e.message);
    return { status: "情報取得エラー", detail: "路線の状態を確認できませんでした" };
  }
}
```

主要路線の英語コード一覧（一部）：

※使いたい路線の一覧を路線マスタで登録しておく

ステップ8: 花粉情報取得処理 🆕

```
function getPollenInfo(cityCode, currentMonth) {
  // 花粉シーズン（2月〜5月）以外はスキップ
  IF currentMonth < 2 OR currentMonth > 5:
    RETURN null

  TRY:
    // tenki.jp の花粉情報API（非公式）
    // 地域コードを都道府県コードに変換（最初の2桁）
    prefCode = String(cityCode).substring(0, 2)
    url = "https://tenki.jp/pollen/" + prefCode + "/"
```

```
    response = UrlFetchApp.fetch(url)
    html = response.getContentText()

    // 今日の花粉飛散レベルを抽出
    // レベル: 1=少ない, 2=やや多い, 3=多い, 4=非常に多い
    levelMatch = html.match(/今日の花粉.+?level-(\d)/s)

    IF !levelMatch:
      RETURN null

    level = parseInt(levelMatch[1])
    levelTexts = ["", "少ない", "やや多い", "多い", "非常に多い"]
    levelEmojis = ["", "😊", "😐", "😷", "🤧"]

    message = levelEmojis[level] + " 花粉：" + levelTexts[level]

    // レベル3以上の場合はアドバイスを追加
    IF level >= 3:
      message += "（マスク・メガネの着用をおすすめします）"

    RETURN message

  CATCH error:
    RETURN null  // エラー時は花粉情報を表示しない
}
```

花粉レベルの定義:

レベル	表示	アイコン	説明
1	少ない	😊	特に対策不要
2	やや多い	😐	敏感な人は注意
3	多い	😷	マスク推奨
4	非常に多い	🤧	万全の対策を

ステップ9: メッセージ組み立て処理（v2.0） 🆕

```
function buildMessage(data) {
  message = data.userName + "さん、おはようございます！\n"
```

```
message += "今日は" + data.date + "(" + data.day + ")です。\n\n"

// 天気情報
message += "【今日の天気】\n"
IF data.weatherData:
  message += data.weatherData.weather
  message += "\n（気温：最高" + data.weatherData.maxTemp + "度"
  message += " / 最低" + data.weatherData.minTemp + "度） "
  message += "\n降水確率：" + data.weatherData.precipitation + "%"
ELSE:
  message += "天気情報を取得できませんでした"

// 天気アラート 🆕
IF data.weatherAlert:
  message += "\n" + data.weatherAlert

message += "\n\n"

// 花粉情報（2-5月のみ） 🆕
IF data.pollenInfo:
  message += "【花粉情報】\n"
  message += data.pollenInfo + "\n\n"

// 鉄道運行情報 🆕
IF data.trainInfo:
  message += "【運行情報】\n"
  message += "🚃 " + data.routeName + "\n"
  message += data.trainInfo.status + "\n"
  IF data.trainInfo.detail:
    message += data.trainInfo.detail + "\n"
  message += "\n"

// ニュース情報
message += "【最新ニュース"
IF data.newsCategory != "一般":
  message += "（" + data.newsCategory + "） "
message += "】\n"
message += data.newsList + "\n\n"
```

```
  message += "今日も一日頑張りましょう！"

  RETURN message
}
```

メッセージ構成の優先順位:

1. 挨拶・日付
2. 天気情報 + アラート
3. 花粉情報（該当月のみ）
4. 運行情報（異常時のみ）
5. ニュース
6. 締めの挨拶

ステップ10: LINE送信処理

(v1.0と同じ)

ステップ11: ログ記録処理（拡張版） 🆕

```
function logResult(logSheet, scheduleData, result) {
  logSheet.appendRow([
    new Date(),                  // 送信日時
    scheduleData.memo,           // 備考
    scheduleData.newsCategory,   // ニュースカテゴリ 🆕
    scheduleData.routeName,      // 路線名 🆕
    result.status,               // 送信結果
    result.weatherAlert ? "有" : "無",  // アラート有無 🆕
    result.trainAlert ? "有" : "無"     // 運行異常有無 🆕
  ])
}
```

---

4. データ構造設計

4.1 スケジュールシート（v2.0）

列	列名	データ型	説明	例
A	有効	TRUE/FALSE	配信ON/OFF	TRUE
B	地域コード	文字列/数値	気象庁の地域コード（6桁）	130000
C	ニュースカテゴリ	文字列	🆕 一般/テクノロジー/ビジネス/スポーツ/エンタメ	テクノロジー
D	路線名	文字列	🆕 鉄道路線名（空欄可）	山手線
E	備考	文字列	スケジュールの目的・メモ	平日用

設定例:

有効	地域コード	ニュースカテゴリ	路線名	備考
TRUE	130000	テクノロジー	山手線	平日・出勤用
TRUE	130000	一般		休日用
FALSE	270000	ビジネス	御堂筋線	大阪出張用（現在停止中）

4.2 ログシート（v2.0）

列	列名	データ型	説明	例
A	送信日時	日時	メッセージ送信日時	2026/1/12 7:00:15
B	備考	文字列	スケジュールの備考	平日・出勤用
C	ニュースカテゴリ	文字列	🆕 使用したカテゴリ	テクノロジー
D	路線名	文字列	🆕 監視した路線	山手線
E	結果	文字列	送信結果	成功
F	アラート有無	文字列	🆕 天気アラートの有無	有
G	運行異常有無	文字列	🆕 鉄道の遅延等の有無	無

ログの活用例:

- アラート発生頻度の分析
- 運行情報の異常発生傾向の把握
- カテゴリ別のニュース配信統計

4.3 路線マスタ（推奨：別シート作成） 🆕

コード内に路線マップを持つ代わりに、スプレッドシートで管理することも推奨します。

列	列名	例
A	路線名	山手線
B	Yahoo!コード	21

---

5. 設定項目

5.1 スクリプトプロパティ（必須設定）

以下の情報を「プロジェクトの設定」→「スクリプトプロパティ」に登録：

プロパティ名	説明	取得方法
LINE_ACCESS_TOKEN	LINEチャネルアクセストークン	LINE Developers コンソールから取得
USER_ID	送信先のLINEユーザーID	LINE公式アカウントのWebhookまたはProfile APIから取得

5.2 トリガー設定

推奨設定:

- トリガータイプ: 時間主導型
- 実行間隔: 日付ベースのタイマー
- 実行時刻: 午前7時〜8時（ユーザーの起床時間に合わせて調整）

複数スケジュール対応パターン:

- 平日用（月〜金 7:00）: ニュース=ビジネス、路線=通勤路線
- 休日用（土日 9:00）: ニュース=エンタメ、路線=なし

→ 曜日判定機能を追加する場合は「拡張2」を参照

---

6. 外部API・データソース仕様

6.1 LINE Messaging API

Profile API（ユーザー名取得）

- **エンドポイント**: `GET https://api.line.me/v2/bot/profile/{userId}`
- **認証**: Bearer token
- **レスポンス例**:

```
{  "displayName": "山田太郎",  "userId": "U1234567890abcdef",  "pictureUrl": "https://...",  "statusMessage": "Hello!"}
```

Push Message API（メッセージ送信）

- **エンドポイント**: `POST https://api.line.me/v2/bot/message/push`
- **認証**: Bearer token
- **リクエスト例**:

```
{  "to": "U1234567890abcdef",  "messages": [    { "type": "text", "text": "メッセージ本文" }  ]}
```

- **成功時レスポンス**: HTTP 200

6.2 気象庁API（拡張版） 🆕

- **エンドポイント**: `GET https://www.jma.go.jp/bosai/forecast/data/forecast/{地域コード}.json`
- **認証**: 不要
- **データ構造**:

```
{
  timeSeries: [
    {
      // [0]: 天気概況
      areas: [{
        weathers: ["晴れ時々曇り 所により雨"],
        // ...
```

```
      }]
    },
    {
      // [1]: 降水確率
      areas: [{
        pops: ["20", "30", "40", "50"], // 6時間ごと
        // ...
      }]
    },
    {
      // [2]: 気温
      areas: [{
        temps: ["5", "15"], // [最低, 最高]
        // ...
      }]
    }
  ]
}
```

取得データ:

- `weathers[0]`: 今日の天気概況
- `pops[0]`: 午前中の降水確率（または `pops[1]` で午後）
- `temps[0]`: 最低気温
- `temps[1]`: 最高気温

6.3 Googleニュース RSS（カテゴリ対応版） 🆕

各カテゴリのRSS URL:

```
const NEWS_URLS = {
  "一般": "https://news.google.com/rss?hl=ja&gl=JP&ceid=JP:ja",
  "テクノロジー": "https://news.google.com/rss/topics/CAAqJggKIiBDQkFTRWdvSUwyMHZNRGRqTVhZU0FtcGhHZ0pLVUNnQVAB?hl=ja&gl=JP&ceid=JP:ja",
  "ビジネス": "https://news.google.com/rss/topics/CAAqJggKIiBDQkFTRWdvSUwyMHZNRGx6TVdZU0FtcGhHZ0pLVUNnQVAB?hl=ja&gl=JP&ceid=JP:ja",
  "スポーツ": "https://news.google.com/rss/topics/CAAqJggKIiBDQkFTRWd
```

```
vSUwyMHZNRFp1ZEdvU0FtcGhHZ0pLVUNnQVAB?hl=ja&gl=JP&ceid=JP:ja",
  "エンタメ": "https://news.google.com/rss/topics/CAAqJggKIiBDQkFTRWdvSUwyMHZNREpxYW5RU0FtcGhHZ0pLVUNnQVAB?hl=ja&gl=JP&ceid=JP:ja"
};
```

6.4 Yahoo!路線情報（スクレイピング） 🆕

- **エンドポイント**: `GET https://transit.yahoo.co.jp/traininfo/detail/{路線コード}/0/`
- **認証**: 不要
- **データ形式**: HTML
- **注意事項**:
  - 非公式APIのため、Yahoo!側の仕様変更で動作しなくなる可能性あり
  - スクレイピングは利用規約を確認すること
  - アクセス頻度に注意（毎朝1回程度なら問題なし）

HTMLパース方法:

```
// 平常運転の判定
html.includes("現在、平常通り運転しています")

// 遅延情報の抽出
html.match(/<div class="trouble">(.+?)<\/div>/s)
```

6.5 tenki.jp 花粉情報（スクレイピング） 🆕

- **エンドポイント**: `GET https://tenki.jp/pollen/{都道府県コード}/`
- **認証**: 不要
- **データ形式**: HTML
- **対応期間**: 2月〜5月（スギ・ヒノキ花粉）

都道府県コード（一部）:

- 13: 東京都
- 14: 神奈川県

- 27: 大阪府
- 23: 愛知県
- 01: 北海道
- 40: 福岡県

HTMLパース方法:

```
// 今日の花粉レベルを抽出
html.match(/今日の花粉.+?level-(\d)/s)
```

---

7. 拡張性の提案

7.1 v2.0で実装済みの拡張

- ✅ **拡張4: 天気アラート機能**
- ✅ **拡張5: ニュースカテゴリ選択**
- ✅ **拡張7: 交通情報（鉄道運行情報）**
- ✅ **拡張（推奨）: 花粉情報**

7.2 今後の拡張候補（基本レベル）

拡張12: 曜日・時刻指定機能

- スケジュールシートに「実行曜日」「実行時刻」列を追加
- 平日のみ、週末のみ、特定曜日のみの配信が可能に
- 複数のトリガーを設定せずに1つのトリガーで管理

拡張13: 気温差アラート

- 前日との気温差が5度以上の場合に通知
- 「昨日より〇度暑い/寒いです」
- 前日の気温をスクリプトプロパティに保存

拡張14: UV指数・紫外線情報

- 気象庁や環境省のUVインデックスを取得
- UV指数が強い場合に日焼け止めを推奨

拡張15: 週間天気予報（週初めのみ）

- 月曜日のみ1週間の天気をまとめて配信
- 週の天気傾向を把握できる

拡張16: カスタムメッセージ挿入

- スケジュールシートに「追加メッセージ」列
- 定型文に加えて個人的なメッセージを追加可能

7.3 今後の拡張候補（中級レベル）

拡張17: 複数路線対応

- 路線名をカンマ区切りで複数指定可能
- 「山手線,中央線,京浜東北線」のように記述
- 異常がある路線のみ表示

拡張18: カレンダー連携

- Googleカレンダーから当日の予定を取得
- 「今日の予定」として表示
- 「9:00 会議」「14:00 打ち合わせ」

拡張19: 通勤時間予測

- Google Maps API で通勤時間を取得
- 「いつもより〇分多くかかりそうです」
- 出発時刻の最適化提案

拡張20: 株価情報

- Yahoo Finance などから特定銘柄の前日終値を取得
- 日経平均、TOPIX、ドル円為替レート
- 投資家向けの朝の情報

拡張21: 送信失敗時のリトライ機能

- LINE API がエラーを返した場合、5分後に再送
- 最大3回までリトライ
- 全て失敗したら管理者にメール通知

拡張22: 祝日判定機能

- Googleカレンダー「日本の祝日」を参照
- 祝日は配信をスキップ（または休日用メッセージ）
- 振替休日にも対応

7.4 今後の拡張候補（高度レベル）

拡張23: LINEリッチメッセージ化

- Flex Messageを使用した視覚的な通知
- 天気アイコン、温度計グラフ、ニュースサムネイル
- ボタンでニュース詳細ページへリンク

拡張24: 双方向コミュニケーション

- LINE Webhook対応
- 「今日の天気は？」と送ると即座に返信
- 「明日の天気は？」にも対応

拡張25: AIによるニュース要約

- Claude API や OpenAI API を使用
- 3つのニュースを1文ずつ要約して表示
- より読みやすく、時短に

拡張26: 画像生成機能

- 天気に応じた画像を自動生成（晴れ=太陽、雨=傘）
- LINE Image Message で配信
- より視覚的で魅力的な通知

拡張27: パーソナライゼーション学習

- ログデータから利用パターンを分析
- よく見る曜日・時間帯を学習
- ユーザーの好みに自動適応

拡張28: 複数チャネル配信

- LINE、Slack、Discord、メールに同時配信

- チャネルごとに最適なフォーマット
- 家族はLINE、仕事はSlackなど

拡張29: 災害情報連携

- 気象庁の特別警報・緊急情報を監視
- 地震、台風、豪雨などの速報
- 緊急時は時間外でも即座に通知

拡張30: ダッシュボード＆分析

- Google Data Studio で可視化
- 配信成功率、アラート発生頻度
- 天気とアラートの相関分析

---

8. 実装時の注意点

8.1 エラーハンドリング（v2.0対応）

天気情報の注意点

- 降水確率データが無い地域がある → デフォルト0%
- 気温データが無い地域がある → 「データなし」表示
- **try-catchで各データ取得を個別に保護** 🆕

ニュース取得の注意点

- 無効なカテゴリが指定された場合は「一般」にフォールバック 🆕
- RSSフィードが空の場合のハンドリング

鉄道運行情報の注意点 🆕

- **路線名が正しくマッピングされない場合のエラー処理**
- Yahoo!のHTML構造変更に備えたエラーハンドリング
- スクレイピング失敗時は「情報取得不可」と表示
- 路線名が空の場合はスキップ

花粉情報の注意点 🆕

- **2月〜5月以外は取得処理自体をスキップ**
- データ取得失敗時はnullを返し、メッセージに含めない
- 地域によって花粉情報が無い場合がある

8.2 パフォーマンス（v2.0対応）

- **1スケジュールあたりの実行時間: 約6〜10秒** 🆕
  - LINE API: 1〜2秒
  - 気象庁API: 1〜2秒
  - GoogleニュースRSS: 1〜2秒
  - Yahoo!路線情報: 1〜2秒 🆕
  - tenki.jp花粉情報: 1〜2秒 🆕
- 複数スケジュール時はAPI呼び出しが増えるため注意
- **GASの実行時間制限（6分）内に収める**
  - 最大約30〜40スケジュール程度が安全圏

8.3 スクレイピングの法的・倫理的注意点 🆕

Yahoo!路線情報

- 利用規約を確認すること
- アクセス頻度を抑える（1日1回程度）
- User-Agentを適切に設定
- HTML構造が変わったら即座に対応

tenki.jp花粉情報

- 同様に利用規約を確認
- 公式APIが提供されている場合はそちらを優先
- 個人利用の範囲内で使用

推奨: 可能であれば公式APIへの移行を検討

- 鉄道運行情報: JR東日本や私鉄各社の公式API（有料の場合あり）
- 花粉情報: 環境省の公式データ（humonghao.cnなど）

8.4 セキュリティ

(v1.0と同じ)

8.5 コード品質（v2.0対応）

改善推奨事項:

1. ✅ `sendLineMessage` 関数の重複削除（v1.0の問題を解決）
2. 🆕 各API呼び出しを関数として完全分離
3. 🆕 定数を設定オブジェクトとして一元管理
4. 🆕 エラーメッセージの統一
5. 🆕 ログ出力の強化（どの処理で時間がかかったかを記録）

推奨コード構造:

```
// 設定定数
const CONFIG = {
  NEWS_CATEGORIES: {...},
  ROUTE_MAP: {...},
  ERROR_MESSAGES: {...}
};

// メイン処理
function main() { ... }

// API呼び出し関数（それぞれ独立）
function getUserDisplayName() { ... }
function getWeatherData() { ... }
function generateWeatherAlert() { ... }
function getNews() { ... }
function getTrainInfo() { ... }
function getPollenInfo() { ... }

// ユーティリティ関数
function convertRouteNameToEnglish() { ... }
function logResult() { ... }
```

---

9. トラブルシューティング

よくある問題と解決策（v2.0対応）

問題	原因	解決策
メッセージが送信されない	トリガー未設定	GASエディタでトリガーを設定
天気が取得できない	地域コードが間違っている	気象庁の公式リストで確認
🆕 降水確率が表示されない	データが存在しない地域	デフォルト0%で表示するようコード修正
🆕 ニュースが「一般」になる	カテゴリ名のスペルミス	スケジュールシートを確認
🆕 運行情報が取得できない	路線名が対応していない	`routeMap` に路線を追加
🆕 花粉情報が表示されない	花粉シーズン外（6-1月）	正常（2-5月のみ表示）
🆕 実行時間が6分を超える	スケジュールが多すぎる	スケジュール数を減らすか、複数トリガーに分割
「ユーザー」と表示される	USER_IDが間違っている	LINE Developersで正しいIDを確認
実行エラーが出る	スクリプトプロパティ未設定	LINE_ACCESS_TOKEN と USER_ID を設定

デバッグ方法 🆕

ログ出力を追加する:

```
Logger.log("天気データ取得開始");
const weatherData = getWeatherData(cityCode);
Logger.log("天気データ: " + JSON.stringify(weatherData));
```

実行ログの確認:

1. GASエディタで「実行数」→「実行ログを表示」
2. エラーメッセージと発生時刻を確認
3. 該当処理のtry-catch内でログ出力

10. 開発の推奨ステップ

Phase 1: v1.0の安定稼働確認 ✅

- 基本機能（天気・ニュース）の動作確認
- 1週間程度の運用で安定性を確認

Phase 2: v2.0への段階的アップグレード

Step 1: データ構造の拡張

1. スケジュールシートに「ニュースカテゴリ」「路線名」列を追加
2. ログシートに対応する列を追加
3. 既存データに影響がないことを確認

Step 2: ニュースカテゴリ機能の追加

1. `getNews()` 関数をカテゴリ対応版に書き換え
2. 手動実行でテスト（各カテゴリを試す）
3. 問題なければ本番運用開始

Step 3: 天気アラート機能の追加

1. 天気データ取得を構造化（降水確率を含める）
2. `generateWeatherAlert()` 関数を追加
3. アラート条件を調整（しきい値の変更）

Step 4: 鉄道運行情報の追加

1. `routeMap` を作成（利用する路線のみ）
2. `getTrainInfo()` 関数を実装
3. 手動実行でYahoo!からの取得を確認
4. HTML構造が変わっていないかテスト

Step 5: 花粉情報の追加

1. `getPollenInfo()` 関数を実装
2. 月判定ロジックを追加
3. 花粉シーズン（2-5月）にテスト

Step 6: 統合テスト

1. 全機能を有効にして手動実行
2. メッセージの見栄えを確認
3. 文字数制限（LINEは最大5000文字）を確認

Step 7: 本番運用開始

1. トリガーを設定
2. 数日間ログを監視
3. エラーがあれば修正

Phase 3: さらなる拡張（オプション）

- 曜日・時刻指定機能
- 複数路線対応
- AIニュース要約
- リッチメッセージ化

---

11. まとめ

v2.0の主な改善点

項目	v1.0	v2.0
天気情報	概況と気温のみ	降水確率追加、アラート機能
ニュース	一般ニュースのみ	5カテゴリから選択可能
交通情報	なし	鉄道運行情報（遅延・運休）
季節情報	なし	花粉情報（2-5月）
メッセージ	固定フォーマット	状況に応じて動的に変化
ログ	基本情報のみ	カテゴリ、アラート有無も記録