スマートフォン上のアプリケーションにおける利用者情報の
取扱いに係る技術的検証等の諸問題に係る実証調査研究

静的解析システムソースコード（バッチ処理）

平成 29 年 2 月 1 日

改訂履歴

版数	日付	変更理由	変更内容	変更者
1.0	H29/2/1	新規作成	1.0 版作成	

目次

1　はじめに

本書は、平成 28 年度の総務省施策である「スマートフォン上のアプリケーションにおける利用者情報の取扱いに係る実証調査研究の請負」における静的解析システムのバッチ処理のソースコードである。

2　ソースファイル一覧

apk アプリ静的解析バッチ処理で必要となるソースファイル一覧を「表 2-1 apk アプリ静的解析バッチ処理ソースファイル一覧」に示す。
処理内容は、静的解析システム仕様書の「5.2 バッチ構成図」の B0001 にあたる。

表 2-1 apk アプリ静的解析バッチ処理ソースファイル一覧

項番	名称	ソースコードファイル名	備考
1	apk アプリ静的解析バッチクラス	ApkAnalysisBatch.java	
2	Android アプリケーション静的解析結果データ格納クラス	AnalysisResponse.java	apk アプリ静的解析バッチクラス から呼び出し
3	データベースクラス	ApkAnalysisDao.java	apk アプリ静的解析バッチクラスから呼び出し
4	メール送信クラス	MailSend.java	apk アプリ静的解析バッチクラスから呼び出し
5	ApacheVelocity のラッパークラス	VelocityWrapper.java	apk アプリ静的解析バッチクラスから呼び出し
6	XML 解析クラス	XmlReader.java	apk アプリ静的解析バッチクラスから呼び出し
7	apk アプリ静的解析バッチシェル	apk_analysis.sh	apk アプリ静的解析バッチ を呼び出すシェルスクリプト

統計情報作成バッチ処理で必要となるソースコードを「表 2-2 統計情報作成バッチ処理ソースファイル一覧」に示す。このうち、データベースクラスについては、「表 2-1 apk アプリ静的解析バッチ処理ソースファイル一覧」で示した データベースクラスと共通である。
処理内容は、静的解析システム仕様書の「5.2 バッチ構成図」の B0002 にあたる。

表 2-2 統計情報作成バッチ処理ソースファイル一覧

項番	名称	ソースコードファイル名	備考
1	統計情報作成バッチクラス	StatisticsBatch.java	

2	データベースクラス	ApkAnalysisDao.java	統計情報作成クラスから呼び出し
3	統計情報作成バッチシェル	apk_statistics.sh	統計情報作成バッチを呼び出すシェルスクリプト

また、返信メールを作成する際のテンプレートファイルと、統計情報を作成する際の設定ファイルを「表 2-3 テンプレート・設定ファイル」に示す。

表 2-3 テンプレート・設定ファイル

項番	名称	ファイル名	備考
1	返信メールテンプレートファイル	analysis_response.vm	Apache Velocity テンプレートファイル
2	統計情報設定ファイル	statistics_association.properties	統計情報の値とセル位置の指定ファイル

3　apk アプリ静的解析処理ソースコード

3.1　apk アプリ静的解析バッチクラス

apk アプリ静的解析バッチクラスのソースコードを以下に示す。

```
package apk_analysis;

import java.io.File;
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.Properties;

import org.apache.logging.log4j.LogManager;
//import java.util.logging.LogManager;
//import java.util.logging.Logger;
import org.apache.logging.log4j.Logger;
import org.apache.velocity.exception.ParseErrorException;
import org.apache.velocity.exception.ResourceNotFoundException;

import apk_analysis.data.AnalysisResponse;
import apk_analysis.db.ApkAnalysisDao;
import apk_analysis.mail.MailSend;
```

```
import apk_analysis.mail.VelocityWrapper;
import apk_analysis.xml.XmlReader;

/**
 * Androidアプリ静的解析依頼バッチクラス
 */
public class ApkAnalysisBatch {

    /**
     *  ロガー
     */
    private static Logger log = LogManager.getLogger(ApkAnalysisBatch.class.getName());

    /**
     * 正常終了ステータス
     */
    private static final int SCCESS_STATUS = 2;

    /**
     * 異常終了ステータス
     */
    private static final int ERROR_STATUS = 3;
```

```
/**
 * Androidアプリ静的解析依頼
 *
 * @param args
 */
public static void main(String[] args) {

    log.info("Androidアプリ解析バッチ Start");
    // DB接続
    ApkAnalysisDao dao = null;
    try {
        Properties props = new Properties();
    // アプリケーション解析情報テンプレートファイル読込
    props.load(new FileInputStream("conf/apk_analysis.properties"));

    // プロセスの最大起動数を取得
    int maxExecCount = Integer.valueOf(props.getProperty("analysis.max_exec_count"));
    // コネクションの確立
        dao = new ApkAnalysisDao(log);

        // 解析依頼実行数非対象とする処理経過時間
        int hourTime = Integer.valueOf(props.getProperty("analysis.max_exec_time"));
```

```
// 制限時間超えの解析リクエストで解析中のものを取得
int timeoutReqNo = dao.getTimeoutRequestNo(hourTime);
// 制限時間超えの解析リクエストが解析中であった場合、異常終了とする
if (timeoutReqNo > 0){
    log.info("制限時間超えの解析中リクエストを異常終了として扱います RequestNo"+timeoutReqNo);

    AnalysisResponse timeoutAnalysisResponse = new AnalysisResponse();
    timeoutAnalysisResponse = dao.getResponseInfo(timeoutReqNo, timeoutAnalysisResponse);
    timeoutAnalysisResponse.setAnalysisStatus(ERROR_STATUS);

    String timeoutAppFilePath = props.getProperty("analysis.filepath") + "/"
            + String.valueOf(timeoutReqNo) + "/" + timeoutAnalysisResponse.getApplicationName();

    postProcess(timeoutAnalysisResponse,  timeoutAppFilePath, dao);
}

// 「解析中」レコード数を取得
int count = dao.getAnalysisRequestNo(hourTime);
// 解析中の件数が上限に達している場合、処理終了
if (count >= maxExecCount) {
    log.info("Android アプリ静的解析中件数が上限に達しているため、処理終了");
    System.exit(0);
}
```

```
// 最古の解析待ち No を取得
int requestNo = dao.getLongestWaitRequestNo();
// 0 の場合は待ち無しのため、終了
if (requestNo == 0) {
    log.info("Android アプリ静的解析待ち無し 正常終了");
    System.exit(0);
}

// 返信用情報取得
AnalysisResponse analysisResponse = new AnalysisResponse();
analysisResponse = dao.getResponseInfo(requestNo, analysisResponse);

// ステータスを実行中に更新
if (!dao.updateStatus(requestNo)) {
    log.error("Android アプリ静的解析待ちステータス更新失敗 異常終了");
    System.exit(1);
}

// Android アプリケーション解析依頼
String appFilePath = props.getProperty("analysis.filepath") + "/"
        + String.valueOf(requestNo) + "/" + analysisResponse.getApplicationName();
String reportPath = props.getProperty("report.filepath")
```

```
            + "/" + String.valueOf(requestNo) + ".xml";
        analysisResponse.setFilePathName(reportPath);

        boolean ret = executeProcess(requestNo, appFilePath, reportPath, props);
        if (ret == false) {
            analysisResponse.setAnalysisStatus(ERROR_STATUS);
            log.error("Android アプリ静的解析依頼エラー 異常");
        } else {
            // XML 読み込み
            try {
                XmlReader xmlReader = new XmlReader(log);

                xmlReader.read(analysisResponse);
            // 結果を DB に登録
            dao.insertResponse(analysisResponse);
            // 正常に終了したように見えるが途中で解析が終わるなど実は異常終了のときエラーを登録するための分岐
            if( !analysisResponse.getResult().equals("04") ){
                analysisResponse.setAnalysisStatus(SCCESS_STATUS);
            }else{
                    analysisResponse.setAnalysisStatus(ERROR_STATUS);
            }
            } catch( ParseErrorException pee ) {
                analysisResponse.setAnalysisStatus(ERROR_STATUS);
```

```
                    // 構文エラー : テンプレートの解析時に問題発生
                    log.error("XML 解析で例外発生",pee);
                }
            }

            postProcess(analysisResponse,  appFilePath, dao);

            log.info("Android アプリ静的解析依頼バッチ 終了");

        } catch( ResourceNotFoundException rnfe ) {
            // テンプレートが見つからない
            log.error("テンプレートが見つかりません",rnfe);
        } catch (Exception e) {
            log.error("想定外のエラー発生",e);
        } finally {
            // データベースクローズ
            if (dao != null) {
                dao.close();
            }
        }
    }

    /**
```

```
 * 指定 Andorid アプリケーションの解析依頼を行う。
 *
 */
private static boolean executeProcess(int requestNo, String appFilePath, String reportPath, Properties props) {

    // 実行コマンド生成
    List<String> command = new ArrayList<String>();
    command.add("java");
    command.add(props.getProperty("command.java.bit"));
    command.add(props.getProperty("command.java.cs"));
    command.add(props.getProperty("command.java.ms"));
    command.add(props.getProperty("command.java.mx"));
    command.add(props.getProperty("command.java.newsize"));
    command.add(props.getProperty("command.java.maxnewsize"));
    command.add(props.getProperty("command.java.ss"));
    command.add(props.getProperty("command.java.log4j"));
    command.add(props.getProperty("command.java.jar.command"));
    command.add(props.getProperty("command.java.jar.file"));
    command.add(appFilePath);
    command.add(props.getProperty("command.java.properties"));
    command.add(reportPath);
    command.add(props.getProperty("command.java.param"));
```

```
        try {
            ProcessBuilder  pb = new ProcessBuilder(command);
            // 標準出力と標準エラー出力パスを統一
            pb.redirectErrorStream(true);
            // 外部プログラムからの出力を本プロセスに統合
            pb.inheritIO();
           File dir = new File(props.getProperty("java.analysis.path"));
           pb.directory(dir);
           Process p = pb.start();
           int preturn = p.waitFor();
            if (preturn != 0) {
                log.error("Androidアプリ解析依頼 エラー 返却値[{}]", preturn);
                return false;
            }
            return true;
        } catch (Exception e) {
            log.error("Androidアプリ解析依頼 例外", e);
            return false;
        }
    }


    private static void postProcess( AnalysisResponse analysisResponse, String appFilePath, ApkAnalysisDao dao) throws ResourceNot
```

```
FoundException, Exception{

        // メールのテンプレートを指定
    VelocityWrapper vw = new VelocityWrapper("templates/analysis_response.vm");
    vw.put("analysis_response", analysisResponse);
    String body = vw.merge();

    // 変換したメール本文
    log.debug(body);

        MailSend mailsend = new MailSend(log);
        mailsend.send(body, analysisResponse);

        // アプリを削除
        File deleteFile = new File(appFilePath);
        log.debug("解析済みのアプリファイルを削除[{}]", appFilePath);
        if (!deleteFile.delete()) {
            log.warn("解析済みのアプリファイルを削除ができませんでした[{}]",appFilePath);
        }

        // ステータスを更新し、メールアドレスをマスキング
        dao.updateAfterExec(analysisResponse);
    }
```

```
}
```

3.2 Androidアプリケーション静的解析結果データ格納クラス

Androidアプリケーション静的解析結果データ格納クラスのソースコードを以下に示す。

```
package apk_analysis.data;

import java.util.ArrayList;
import java.util.List;

/**
 * Androidアプリケーション静的解析結果データ格納クラス
 */
public class AnalysisResponse {

    /**
     * 受付No
     */
    private Integer requestNo;

    /**
     * 申請受付年
     */
    private int requestYear;
```

```
    /**
     * 申請受付月
     */
    private int requestMonth;

    /**
     * 申請受付日
     */
    private int requestDay;

    /**
     * 申請受付時
     */
    private int requestHour;

    /**
     * 申請受付分
     */
    private int requestMinute;

    /**
     * 解析結果ステータス
     */
```

```
    private int analysisStatus;

    /**
     * アプリケーション名
     */
    private String applicationName;

    /**
     * メールアドレス
     */
    private String mailaddress;

    /**
     * 解析結果ファイルパス
     */
    private String filePathName;

    /**
     * 解析結果
     */
    private String result;

    /**
```

```
     *  apk ファイル名
     */
    private String apkFileName;

    /**
     * apk ファイルを一意に示す ID。SHA256 ハッシュ
     */
    private String apkHash;

    /**
     * バージョン
     */
    private String apkVersion;

    /**
     * apk ファイルの生成日付
     */
    private String apkDate;

    /**
     * AndroidManifest.xml で定義された uses-permission 要素の android:name 属性
     */
    private List<String> permission;
```

```
/**
 * 動的クラスローダーを実行しているメソッド名
 */
private List<String> classLoaderExecute;

/**
 * 内部で Script を実行しているメソッド名
 */
private List<String> scriptExecute;

/**
 * 利用者情報の送信
 */
private Integer detailResult;

/**
 * 利用者情報詳細結果
 */
private List<Integer> caseList;


public Integer getRequestNo() {
```

```
        return requestNo;
    }

    public void setRequestNo(Integer requestNo) {
        this.requestNo = requestNo;
    }



    public int getRequestYear() {
        return requestYear;
    }

    public void setRequestYear(int requestYear) {
        this.requestYear = requestYear;
    }

    public int getRequestMonth() {
        return requestMonth;
    }

    public void setRequestMonth(int requestMonth) {
        this.requestMonth = requestMonth;
    }
```

```
public int getRequestDay() {
    return requestDay;
}

public void setRequestDay(int requestDay) {
    this.requestDay = requestDay;
}

public int getRequestHour() {
    return requestHour;
}

public void setRequestHour(int requestHour) {
    this.requestHour = requestHour;
}

public int getRequestMinute() {
    return requestMinute;
}

public void setRequestMinute(int requestMinute) {
    this.requestMinute = requestMinute;
```

```
    }

    public int getAnalysisStatus() {
        return analysisStatus;
    }

    public void setAnalysisStatus(int analysisStatus) {
        this.analysisStatus = analysisStatus;
    }

    public String getMailaddress() {
        return mailaddress;
    }

    public void setMailaddress(String mailaddress) {
        this.mailaddress = mailaddress;
    }

    public String getApplicationName() {
        return applicationName;
    }

    public void setApplicationName(String applicationName) {
```

```
        this.applicationName = applicationName;
    }

    public Integer getDetailResult() {
        return detailResult;
    }

    public void setDetailResult(Integer detailResult) {
        this.detailResult = detailResult;
    }

    public String getFilePathName() {
        return filePathName;
    }

    public void setFilePathName(String filePathName) {
        this.filePathName = filePathName;
    }

    public String getResult() {
        return result;
    }
```

```
    public void setResult(String result) {
        this.result = result;
    }

    public String getApkFileName() {
        return apkFileName;
    }

    public void setApkFileName(String apkFileName) {
        this.apkFileName = apkFileName;
    }

    public String getApkHash() {
        return apkHash;
    }

    public void setApkHash(String apkHash) {
        this.apkHash = apkHash;
    }

    public String getApkVersion() {
        return apkVersion;
    }
```

```
public void setApkVersion(String apkVersion) {
    this.apkVersion = apkVersion;
}

public String getApkDate() {
    return apkDate;
}

public void setApkDate(String apkDate) {
    this.apkDate = apkDate;
}

public List<String> getPermission() {
    return permission;
}

public void setPermission(ArrayList<String> arrayList) {
    this.permission = arrayList;
}

public List<String> getClassLoaderExecute() {
    return classLoaderExecute;
```

```
    }

    public void setClassLoaderExecute(ArrayList<String> arrayList) {
        this.classLoaderExecute = arrayList;
    }

    public List<String> getScriptExecute() {
        return scriptExecute;
    }

    public void setScriptExecute(ArrayList<String> arrayList) {
        this.scriptExecute = arrayList;
    }

    public List<Integer> getCaseList() {
        return caseList;
    }

    public void setCaseList(List<Integer> caseList) {
        this.caseList = caseList;
    }

}
```

3.1　データベースクラス

データベースクラスのソースコードを以下に示す。

```
package apk_analysis.db;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.util.Calendar;
import java.util.HashMap;
import java.util.List;
import java.util.Properties;

import org.apache.logging.log4j.Logger;
```

```
import apk_analysis.data.AnalysisResponse;

/**
 * DB クラス
 */
public class ApkAnalysisDao {
    /**
     * コネクション
     */
    private Connection con;

    /**
     * ロガー
     */
    private Logger log;

    /**
     * プロパティ
     */
    private Properties props;

    /**
     * /**
```

```
 * インスタンス生成時にコネクションを確立する。
 * @throws SQLException DB 接続に失敗した場合スローします。
 * @throws ClassNotFoundException ドライバのロードに失敗した場合スローします。
 * @throws IOException
 * @throws FileNotFoundException
 */
public ApkAnalysisDao(Logger log) throws SQLException,
        ClassNotFoundException, FileNotFoundException, IOException {
// ロガー設定
this.log = log;
this.props = new Properties();
this.log.info("ApkAnalysisDao Contract Start");
// DB 情報テンプレートファイル読込
this.props.load(new FileInputStream("conf/apk_analysis.properties"));
    // ドライバのロード
    Class.forName(this.props.getProperty("database.driver"));
    // コネクションの確立 (DB に接続)
    this.con = DriverManager.getConnection(this.props.getProperty("database.url"),
         this.props.getProperty("database.username"),
         this.props.getProperty("database.password"));
this.log.info("ApkAnalysisDao Contract End");
}
```

```
    /**
     * コネクションを close する。
     *
     */
    public void close() {
        try {
            if (this.con != null) {
                this.con.close();
            }
        } catch (SQLException e) {
         log.error("コネクションクローズ例外[{}]", e);
        }
    }


    /**
     * 解析タイムアウトのリクエスト No を取得
     * @return タイムアウトしたリクエスト No
     * @throws SQLException SQL 実行に失敗した場合にスローします。
     */
    public int getTimeoutRequestNo(int hourTime) throws SQLException {
    PreparedStatement pstmt = null;
        int requestNo = 0;
```

```
    try {
        pstmt = this.con.prepareStatement(String.format(this.props.getProperty("sql.select.timeout_request"),hourTime));
        this.log.info("時間切れリクエストNo取得SQL[{}]", pstmt.toString());
        ResultSet result=pstmt.executeQuery();
        result.next();
        if (result.getString(1) != null && result.getString(1).length() != 0){
         requestNo = result.getInt(1);
        }
    } finally {
        this.log.info("取得時間切れリクエストNo[{}]", requestNo);
     try {
            if (pstmt != null) {
                pstmt.close();
            }
        } catch (SQLException e) {
         throw e;
        }
    }
    return requestNo;
}

/**
```

```
     * 解析中の件数を取得
     * @return 解析中の件数
     * @throws SQLException SQL 実行に失敗した場合にスローします。
     */
    public int getAnalysisRequestNo(int hourTime) throws SQLException {
    PreparedStatement pstmt = null;
        int count = 0;

        try {
            pstmt = this.con.prepareStatement(String.format(this.props.getProperty("sql.select.exec_status_count"),hourTime));
            this.log.info("解析中の件数を取得 SQL[{}]", pstmt.toString());
            ResultSet result=pstmt.executeQuery();
            result.next();
            count = result.getInt(1);
        } finally {
            this.log.info("解析中の件数[{}]", count);
        try {
                if (pstmt != null) {
                    pstmt.close();
                }
            } catch (SQLException e) {
             throw e;
            }
```

```
        }
        return count;
    }

    /**
     * リクエストNoを取得
     * @return 一番長く待っているリクエストNo
     * @throws SQLException SQL 実行に失敗した場合にスローします。
     */
    public int getLongestWaitRequestNo() throws SQLException {
    PreparedStatement pstmt = null;
        int requestNo = 0;

        try {
            pstmt = this.con.prepareStatement(this.props.getProperty("sql.select.wait_status"));
            this.log.info("リクエストNo取得SQL[{}]", pstmt.toString());
            ResultSet result=pstmt.executeQuery();
            result.next();
            if (result.getString(1) != null && result.getString(1).length() != 0){
             requestNo = result.getInt(1);
            }
        } finally {
            this.log.info("取得リクエストNo[{}]", requestNo);
```

```
        try {
                if (pstmt != null) {
                    pstmt.close();
                }
            } catch (SQLException e) {
             throw e;
            }
        }
        return requestNo;
    }

    /**
     * リクエストNoに紐づくアプリケーション名称を取得する
     * @return アプリケーション名
     * @throws SQLException SQL 実行に失敗した場合にスローします。
     */
    public AnalysisResponse getResponseInfo(int requestNo, AnalysisResponse analysisResponse) throws SQLException {
    PreparedStatement pstmt = null;

        try {
            pstmt = this.con.prepareStatement(this.props.getProperty("sql.select.application_name"));
            pstmt.setInt(1, requestNo);
            this.log.info("返信情報取得取得 SQL[{}]", pstmt.toString());
```

```
ResultSet result = pstmt.executeQuery();

if(result.next()){
// リクエストNo
analysisResponse.setRequestNo(requestNo);
// 受付年
analysisResponse.setRequestYear(result.getInt(1));
// 受付月
analysisResponse.setRequestMonth(result.getInt(2));
// 受付日
analysisResponse.setRequestDay(result.getInt(3));
// 受付時
analysisResponse.setRequestHour(result.getInt(4));
// 受付分
analysisResponse.setRequestMinute(result.getInt(5));
// アプリケーション名
analysisResponse.setApplicationName(result.getString(6));
// メールアドレス
analysisResponse.setMailaddress(result.getString(7));
}else{
    this.log.debug("返信情報が存在しません");
    this.updateErrorStatus(requestNo);
    throw new SQLException("返信情報が存在しません", "", -100);
```

```
            }
        } finally {
            this.log.debug("返信情報 受付年[{}]月[{}]日[{}]", analysisResponse.getRequestYear(),
                                analysisResponse.getRequestMonth(), analysisResponse.getRequestDay());
            this.log.debug("返信情報 受付時[{}]", analysisResponse.getRequestHour());
            this.log.debug("返信情報 受付分[{}]", analysisResponse.getRequestMinute());
            this.log.debug("返信情報 アプリケーション名[{}]", analysisResponse.getApplicationName());

            try {
                if (pstmt != null) {
                    pstmt.close();
                }
            } catch (SQLException e) {
             throw e;
            }
        }
        return analysisResponse;
    }

    /**
     * Android アプリ静的解析を行うリクエスト No の対象ステータスを解析中に更新
     * @param requestNo リクエスト No
     * @return true 成功 ,false 失敗
```

```
     * @throws SQLException SQL 実行に失敗した場合にスローします。
     */
    public boolean updateStatus(int requestNo) throws SQLException {
        PreparedStatement pstmt = null;
        int count = 0;

        try {
            pstmt = this.con.prepareStatement(String.format(this.props.getProperty("sql.update.status"), "analysis_starttime=CURRE
NT_TIMESTAMP"));
            pstmt.setInt(1, 1);
            pstmt.setInt(2, requestNo);
            // SQL 文を実行します。
            this.log.info("Android アプリ静的解析を行うリクエスト No の対象ステータスを解析中に更新 SQL[{}]", pstmt.toString());
            count = pstmt.executeUpdate();
        } finally {
            try {
                if (pstmt != null) {
                    pstmt.close();
                }
            } catch (SQLException e) {
                throw e;
            }
        }
```

```
        if (count < 0) {
        // 更新失敗
        log.error("解析結果情報更新失敗");
        return false;
        }
        return true;
    }

    /**
     * Android アプリ静的解析結果を登録
     * @param analysisResponse 解析結果情報
     * @return true 成功 ,false 失敗
     * @throws SQLException SQL 実行に失敗した場合にスローします。
     */
    public boolean insertResponse(AnalysisResponse analysisResponse) throws SQLException {
        PreparedStatement pstmt = null;
        int count = 0;

        try {
            // 解析結果情報
            pstmt = this.con.prepareStatement(this.props.getProperty("sql.insert.reponse"));
            pstmt.setInt(1, analysisResponse.getRequestNo());
            pstmt.setString(2, analysisResponse.getResult());
```

```
            pstmt.setInt(3, analysisResponse.getPermission().size());
            pstmt.setInt(4, analysisResponse.getClassLoaderExecute().size());
            pstmt.setInt(5, analysisResponse.getScriptExecute().size());
            pstmt.setInt(6, analysisResponse.getDetailResult());
            List<Integer> caseList = analysisResponse.getCaseList();
            for (int i=0; i < caseList.size(); i++) {
             if (caseList.get(i) == null){
                 pstmt.setNull (i + 7, java.sql.Types.INTEGER);
             } else {
                 pstmt.setInt(i + 7, caseList.get(i));
             }
            }
            // SQL 文を実行します。
            this.log.info("Android アプリ静的解析結果を登録 SQL[{}]", pstmt.toString());
            count = pstmt.executeUpdate();

            if (count < 0) {
             // 登録失敗
             log.error("解析結果情報登録失敗");
             return false;
            }
        } finally {
            try {
```

```
            if (pstmt != null) {
                pstmt.close();
            }
        } catch (SQLException e) {
         throw e;
        }
    }
    return true;
}



/**
 * Android アプリ静的解析を行うリクエスト No の対象ステータスを異常に更新
 * @param requestNo リクエスト No
 * @return true 成功 ,false 失敗
 * @throws SQLException SQL 実行に失敗した場合にスローします。
 */
public boolean updateErrorStatus(int requestNo) throws SQLException {
    PreparedStatement pstmt = null;
    int count = 0;

    try {
```

```
            // Androidアプリ静的解析状態更新
            pstmt = this.con.prepareStatement(String.format(this.props.getProperty("sql.update.status"), "analysis_endtime=CURRENT
_TIMESTAMP"));
            // バインド変数に値を設定します。
            pstmt.setInt(1, 3);
            pstmt.setInt(2, requestNo);
            // SQL 文を実行します。
            this.log.info("Androidアプリ静的解析を行うリクエストNoの対象ステータスを異常に更新SQL[{}]", pstmt.toString());
            count = pstmt.executeUpdate();
            if (count < 0) {
             // 更新失敗
             log.error("Androidアプリ静的解析状態更新失敗");
             return false;
            }
            pstmt.close();
        } finally {
            try {
                if (pstmt != null) {
                    pstmt.close();
                }
            } catch (SQLException e) {
             throw e;
           }
```

```
        }
        return true;
    }

    /**
     * Androidアプリ静的解析を行うリクエストNoの対象ステータスを解析済に更新
     * @param analysisResponse 解析結果情報
     * @return true 成功 ,false 失敗
     * @throws SQLException SQL 実行に失敗した場合にスローします。
     */
    public boolean updateAfterExec(AnalysisResponse analysisResponse) throws SQLException {
        PreparedStatement pstmt = null;
        int count = 0;

        try {
            // Androidアプリ静的解析状態更新
            pstmt = this.con.prepareStatement(String.format(this.props.getProperty("sql.update.status"), "analysis_endtime=CURRENT
_TIMESTAMP"));
            pstmt.setInt(1, analysisResponse.getAnalysisStatus());
            pstmt.setInt(2, analysisResponse.getRequestNo());
            // SQL 文を実行します。
            this.log.info("Androidアプリ静的解析状態更新SQL[{}]", pstmt.toString());
            count = pstmt.executeUpdate();
```

```
            if (count < 0) {
             // 更新失敗
             log.error("Androidアプリ静的解析状態更新失敗");
             return false;
            }
            pstmt.close();
            // Androidアプリ解析結果返信情報更新
            pstmt = this.con.prepareStatement(this.props.getProperty("sql.update.reponse_info"));
            String mailAddress = analysisResponse.getMailaddress();
            String work = mailAddress.substring(mailAddress.indexOf('@'));
            pstmt.setString(1, "******" + work);
            pstmt.setInt(2, analysisResponse.getRequestNo());
            this.log.info("Androidアプリ解析結果返信情報更新 SQL[{}]", pstmt.toString());
            count = pstmt.executeUpdate();
            if (count < 0) {
             // 更新失敗
             log.error("Androidアプリ解析結果返信情報更新失敗");
             return false;
            }
        } finally {
            try {
                if (pstmt != null) {
                    pstmt.close();
```

```
            }
        } catch (SQLException e) {
         throw e;
        }
    }
    return true;
}

/**
 * 統計情報取得
 * @param requestNo リクエストNo
 * @return true 成功 ,false 失敗
 * @throws SQLException SQL 実行に失敗した場合にスローします。
 * @throws InterruptedException
 */
public HashMap<String,Integer> selectStatisticsExec(int year,int month) throws SQLException, InterruptedException {
    PreparedStatement pstmt = null;
    HashMap<String,Integer> statistics = new HashMap<String,Integer>();

    try {
     Calendar cal = Calendar.getInstance();
     // 統計対象月の開始時刻をミリ秒算出
     cal.set(year, month-1,1,0,0,0);
```

```
Timestamp startTimestamp = new Timestamp(cal.getTimeInMillis());
startTimestamp.setNanos(0);
// 統計対象月の終了時刻をミリ秒算出
cal.set(year, month,1,0,0,0);
Timestamp endTimestamp = new Timestamp(cal.getTimeInMillis());
endTimestamp.setNanos(0);

    pstmt = this.con.prepareStatement(this.props.getProperty("sql.select.statistics"));
    pstmt.setTimestamp(1, startTimestamp);
    pstmt.setTimestamp(2, endTimestamp);

    this.log.info("統計情報取得 SQL[{}]", pstmt.toString());
    ResultSet result = pstmt.executeQuery();
    ResultSetMetaData rsmd= result.getMetaData();
    result.next();

    for (int i = 1; i <= rsmd.getColumnCount(); i++){
     statistics.put(rsmd.getColumnName(i),result.getInt(i));
     log.debug("{}[{}]",rsmd.getColumnName(i),result.getInt(i));
    }
    result.close();
    pstmt.close();
```

```
            pstmt = this.con.prepareStatement(this.props.getProperty("sql.select.statistics_family"));
            pstmt.setTimestamp(1, startTimestamp);
            pstmt.setTimestamp(2, endTimestamp);
            this.log.info("統計情報取得(小カテゴリ)SQL[{}]", pstmt.toString());
            result = pstmt.executeQuery();
            rsmd= result.getMetaData();
            result.next();

            for (int i = 1; i <= rsmd.getColumnCount(); i++){
             statistics.put(rsmd.getColumnName(i),result.getInt(i));
             log.debug("{}[{}]",rsmd.getColumnName(i),result.getInt(i));
            }

        } finally {
            try {
                if (pstmt != null) {
                    pstmt.close();
                }
            } catch (SQLException e) {
             throw e;
            }
        }
        return statistics;
```

```
    }
}
```

3.2　メール送信クラス

メール送信クラスのソースコードを以下に示す。

```
package apk_analysis.mail;

import java.io.FileInputStream;
import java.util.Date;
import java.util.Properties;

import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Multipart;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeBodyPart;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeMultipart;

import org.apache.logging.log4j.Logger;

import apk_analysis.data.AnalysisResponse;
```

```
/**
 * メール送信
 *
 */
public class MailSend {

    /**
     * 正常終了ステータス
     */
    private static final int SCCESS_STATUS = 2;

    /**
     * 異常終了ステータス
     */
    private static final int ERROR_STATUS = 3;

    /**
     * ロガー
     */
    private Logger log;

    /**
     * コンストラクタ
```

```
 */
public MailSend(Logger log) {
    this.log = log;
}

public void send(String body, AnalysisResponse analysisResponse) throws Exception{

    Properties mprops = new Properties();
    mprops.load(new FileInputStream("conf/apk_analysis.properties"));

    // subjectには件数、sendMsgにはメール本文がセットされている
    // SMTPホストの設定
    Properties props = System.getProperties();
    props.put("mail.smtp.host", mprops.getProperty("mail.smtp.host"));

    // 接続時のタイムアウト
    String connectionTimeout = mprops.getProperty("mail.smtp.connectiontimeout");
    props.put("mail.smtp.connectiontimeout", new Integer(connectionTimeout));

    // 送受信時のタイムアウト
    String timeout = mprops.getProperty("mail.smtp.timeout");
    props.put("mail.smtp.timeout", new Integer(timeout));
```

```
        // Sessionオブジェクトの取得
        Session session = Session.getDefaultInstance(props, null);
        MimeMessage msg = new MimeMessage(session);
        try {

            msg.setFrom(new InternetAddress(mprops.getProperty("mail.from")));
            msg.setSender(new InternetAddress(mprops.getProperty("mail.sender")));
            msg.setRecipient(Message.RecipientType.TO, new InternetAddress(analysisResponse.getMailaddress()));

            if ( ( ERROR_STATUS == analysisResponse.getAnalysisStatus()) && (mprops.getProperty("mail.admin") != null) ){
                this.log.info("エラーメールとして管理者宛にも送信します Bcc:" + mprops.getProperty("mail.admin"));
                msg.setRecipient(Message.RecipientType.BCC, new InternetAddress(mprops.getProperty("mail.admin")));
            }
            // 件名の設定
            msg.setSubject(analysisResponse.getRequestNo()  + "の解析結果 (JSSEC Android apk Analysis service) ", mprops.getProper
ty("mail.charset"));
        } catch (MessagingException e) {
            e.printStackTrace();
        }

        // 日付の設定
        msg.setSentDate(new Date());
```

```
        // html mail
        Multipart alternativePart = new MimeMultipart("alternative");
        MimeBodyPart htmlBodyPart = new MimeBodyPart();
        htmlBodyPart.setText(body, mprops.getProperty("mail.charset"), "html");
        htmlBodyPart.setHeader("Content-Transfer-Encoding", mprops.getProperty("mail.encoding"));
        alternativePart.addBodyPart(htmlBodyPart);

        // set alternative
        msg.setContent(alternativePart);

//      msg.writeTo(System.out);  // メール内容表示
        Transport.send(msg);
    }
}
```

3.3 ApacheVelocity のラッパークラス

メール送信の際に利用する ApacheVelocity のラッパークラスのソースコードを以下に示す。

```
package apk_analysis.mail;

import java.io.IOException;
import java.io.StringWriter;
import java.util.Properties;
```

```
import org.apache.velocity.Template;
import org.apache.velocity.VelocityContext;
import org.apache.velocity.app.VelocityEngine;
import org.apache.velocity.exception.MethodInvocationException;
import org.apache.velocity.exception.ParseErrorException;
import org.apache.velocity.exception.ResourceNotFoundException;

public class VelocityWrapper {

    /** テンプレート */
    private Template template = null;

    /** テンプレート変換用データオブジェクト */
    private VelocityContext context = new VelocityContext();

    /** Velocity エンジン */
    private VelocityEngine engine = new VelocityEngine();

    /** コンストラクタ */
    public VelocityWrapper(String templateFileName) throws IOException,Exception {

    // velocity.properties による VelocityEngine の初期化
```

```
    Properties props = new Properties();
    // 定義読込
    props.put("input.encoding","utf-8");
    props.put("output.encoding", "utf-8");
    engine.init(props);

        // テンプレートの取得
        template = engine.getTemplate(templateFileName);
    }

    /**
     * テンプレート変換処理用のオブジェクトを格納する
     * @param key    格納キー
     * @param value  格納するデータオブジェクト
     */
    public void put(String key, Object value) {
        context.put(key, value);
    }

    /**
     * テンプレートとキーを変換
     * @return
     * @throws ResourceNotFoundException
```

```
     * @throws ParseErrorException
     * @throws MethodInvocationException
     * @throws Exception
     */
    public String merge()
        throws ResourceNotFoundException,ParseErrorException,
            MethodInvocationException,Exception{
        StringWriter sw = new StringWriter();
        template.merge( context , sw );
        return sw.toString();
    }
}
```

3.4 XML 解析クラス

XML 解析クラスのソースコードを以下に示す。

```
package apk_analysis.xml;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Properties;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;

import org.apache.logging.log4j.Logger;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
```

```
import org.xml.sax.SAXException;

import apk_analysis.data.AnalysisResponse;



/**
 * Androidアプリ静的解析結果レポートXMLファイル解析
 *
 */
public class XmlReader {

    /**
     * ロガー
     */
    private Logger log;

    /**
     * コンストラクタ
     */
    public XmlReader(Logger log) {
        this.log = log;
    }
```

```
    /**
     * Android アプリ静的解析結果レポート XML ファイルを読み込み、引数にオブジェクトに格納
     * @param analysisResponse
     * @throws SAXException
     * @throws IOException
     * @throws ParserConfigurationException
     */
    public void read(AnalysisResponse analysisResponse) throws SAXException, IOException,  XPathExpressionException,ParserConfigur
ationException {
        DocumentBuilder builder = DocumentBuilderFactory.newInstance()
                                                    .newDocumentBuilder();
        Document doc = builder.parse( new File(analysisResponse.getFilePathName()) );
        XPathFactory factory = XPathFactory.newInstance();
        XPath xpath = factory.newXPath();

        Properties props = new Properties();
        props.load(new FileInputStream("conf/apk_analysis.properties"));
        String xpathKey;

        // 解析結果
        xpathKey = props.getProperty("xpath.execinfo.result");
        analysisResponse.setResult(xpath.evaluate(xpathKey, doc));
        log.debug("解析結果[{}] 値[{}]", xpathKey, analysisResponse.getResult());
```

```
if(2 < analysisResponse.getResult().length()){
 log.debug("解析結果の値に入る文字列が長すぎる異常が発生しているため、04 として処理します");
 analysisResponse.setResult("04");
}

// apk ファイル名
xpathKey = props.getProperty("xpath.apkinfo.apkfilename");
analysisResponse.setApkFileName(xpath.evaluate(xpathKey, doc));
log.debug("apk ファイル名[{}] 値[{}]", xpathKey, analysisResponse.getApkFileName());

// apk ファイルを一意に示す ID。SHA256 ハッシュ
xpathKey = props.getProperty("xpath.apkinfo.apkhash");
analysisResponse.setApkHash(xpath.evaluate(xpathKey, doc));
log.debug("SHA256 ハッシュ[{}] 値[{}]", xpathKey, analysisResponse.getApkHash());

// バージョン
xpathKey = props.getProperty("xpath.apkinfo.version");
analysisResponse.setApkVersion(xpath.evaluate(xpathKey, doc));
log.debug("バージョン[{}] 値[{}]", xpathKey, analysisResponse.getApkVersion());

// apk ファイルの生成日付
xpathKey = props.getProperty("xpath.apkinfo.apkdate");
analysisResponse.setApkDate(xpath.evaluate(xpathKey, doc));
```

```
        log.debug("apk ファイルの生成日付[{}] 値[{}]", xpathKey, analysisResponse.getApkDate());

        // AndroidManifest.xml で定義された uses-permission 要素の android:name 属性
        xpathKey = props.getProperty("xpath.apkinfo.permission");
        if (xpath.evaluate(xpathKey, doc) == null || (xpath.evaluate(xpathKey, doc)).length() == 0) {
        analysisResponse.setPermission( new ArrayList<String>());
        } else {
        NodeList permissionList = (NodeList)xpath.evaluate(xpathKey, doc,XPathConstants.NODESET);
        String workPermission;
        ArrayList<String> setPermission = new ArrayList<String>();
        for(int i=0; i < permissionList.getLength(); i++){
                workPermission = permissionList.item(i).getTextContent();
                setPermission.add(workPermission);
                log.debug("AndroidManifest.xml で定義された uses-permission 要素の android:name 属性[{}] 添字[{}] 値[{}]", xpathKey,
i, workPermission);
            }
        analysisResponse.setPermission(setPermission);

        }

        // 動的クラスローダーを実行しているメソッド名
        xpathKey = props.getProperty("xpath.apkinfo.classloaderexecute");
        if (xpath.evaluate(xpathKey, doc) == null || (xpath.evaluate(xpathKey, doc)).length() == 0) {
```

```
        analysisResponse.setClassLoaderExecute(new ArrayList<String>());
        } else {
        NodeList classLoaderExecuteList = (NodeList)xpath.evaluate(xpathKey, doc,XPathConstants.NODESET);
        String workClassLoaderExecute;
        ArrayList<String> setClassLoaderExecute = new ArrayList<String>();
        for(int i=0; i < classLoaderExecuteList.getLength(); i++){
                workClassLoaderExecute = classLoaderExecuteList.item(i).getTextContent();
                setClassLoaderExecute.add(workClassLoaderExecute);
             log.debug("動的クラスローダーを実行しているメソッド名[{}] 添字[{}] 値[{}]", xpathKey, i, workClassLoaderExecute);
            }
        analysisResponse.setClassLoaderExecute(setClassLoaderExecute);
        }


        // 内部で Script を実行しているメソッド名
        xpathKey = props.getProperty("xpath.apkinfo.scriptexecute");
        if (xpath.evaluate(xpathKey, doc) == null || (xpath.evaluate(xpathKey, doc)).length() == 0) {
        analysisResponse.setScriptExecute(new ArrayList<String>());
        } else {
        NodeList scriptExecuteList = (NodeList)xpath.evaluate(xpathKey, doc,XPathConstants.NODESET);
        String workScriptExecute;
        ArrayList<String> setScriptExecute = new ArrayList<String>();
        for(int i=0; i < scriptExecuteList.getLength(); i++){
```

```
                workScriptExecute = scriptExecuteList.item(i).getTextContent();
                setScriptExecute.add(workScriptExecute);
             log.debug("内部で Script を実行しているメソッド名[{}] 添字[{}] 値[{}]", xpathKey, i, workScriptExecute);
            }
         analysisResponse.setScriptExecute(setScriptExecute);
        }

        // 利用者情報の送信
        xpathKey = props.getProperty("xpath.evalinfo.result");
        if (xpath.evaluate(xpathKey, doc) == null || (xpath.evaluate(xpathKey, doc)).length() == 0) {
         analysisResponse.setDetailResult(0);
        } else {
         analysisResponse.setDetailResult(1);
        }
    log.debug("利用者情報の送信[{}] 値[{}]", xpathKey, analysisResponse.getDetailResult());

    // 指定数分解析を行う
    String casePath = new String();
    xpathKey = props.getProperty("xpath.evalinfo.result.case");
    String countWork = props.getProperty("xpath.evalinfo.result.case.count");
    int caseCount = Integer.valueOf(countWork);
    ArrayList<Integer> workCase = new ArrayList<Integer>();
    for (int i=1; i <= caseCount; i++) {
```

```
        casePath = String.format("%s%03d", xpathKey, i);

            if (xpath.evaluate(casePath, doc) == null || (xpath.evaluate(casePath, doc)).length() == 0) {
            workCase.add(null);
            } else {
            String detail = xpath.evaluate(casePath, doc);
            if (Integer.valueOf(detail) == 1) {
                workCase.add(1);
            } else {
                workCase.add(0);
            }
            log.debug("CASE 情報[{}] 値[{}]", casePath, detail);
            }
    }
    analysisResponse.setCaseList(workCase);

    }
}
```

3.5　apk アプリ静的解析バッチシェル

apk アプリ静的解析バッチシェルのソースコードを以下に示す。

```
#!/bin/bash

LANG=ja_JP.UTF8
export LANG

cd /home/analyzer/bin;

if [ -d /var/data/request ]; then
        java  -Dlog4j.configurationFile="./conf/log4j2.xml" -jar apk_analysis.jar
fi
```

4　統計情報作成ソースコード

4.1　統計情報作成クラス

統計情報作成クラスのソースコードを以下に示す。

```
package apk_analysis;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.Properties;

import org.apache.commons.lang.math.NumberUtils;
import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
```

```
import org.apache.poi.ss.usermodel.WorkbookFactory;
import org.apache.poi.ss.util.CellReference;

import apk_analysis.db.ApkAnalysisDao;

public class StatisticsBatch {
    /**
     *  ロガー
     */
    private static Logger log = LogManager.getLogger(StatisticsBatch.class.getName());

    /**
     * 機能名: 統計情報生成
     *
     * @param args 1: 統計年 2: 統計月
     */
    public static void main(String[] args) {

        // パラメータ確認
        if (args.length != 2 || !NumberUtils.isNumber(args[0]) || !NumberUtils.isNumber(args[1])) {
            log.warn("統計情報生成 起動パラメータ不正");
            System.exit(1);
        }
```

```
        int year = Integer.valueOf(args[0]);
        int month = Integer.valueOf(args[1]);
        // DB 接続
        ApkAnalysisDao dao = null;

        try {
            // コネクションの確立
            dao = new ApkAnalysisDao(log);

            // 統計情報出力位置設定ファイル読み込み
            Properties props = new Properties();
            props.load(new FileInputStream("conf/statistics_association.properties"));

            // 統計データ取得(取得年,取得月)
            HashMap<String, Integer> statisticsData = dao.selectStatisticsExec(year, month);

            //共通インターフェースを扱える、WorkbookFactory で読み込む
            Workbook wb = WorkbookFactory.create(new FileInputStream("templates/statistics_template.xlsx"));
            // シートを選択
            Sheet sheet = wb.getSheetAt(0);
            String cellArea;
            CellReference reference;
            Row row;
```

```
        Cell cell;
        // 統計情報
        for(Map.Entry<String, Integer> record: statisticsData.entrySet()){
         // レコードの位置を統計情報設定ファイルから取得
         cellArea = props.getProperty(record.getKey());
         // 統計情報設定ファイルに未設定のキーの場合はスキップ
         if (cellArea != null && cellArea.length() > 0 ){
             // 出力Cell位置を取得
             reference = new CellReference(cellArea);
             row = sheet.getRow(reference.getRow());
                 cell = row.getCell(reference.getCol());
                 // 指定位置に値を出力
                 cell.setCellValue(record.getValue());
         }
        }

        // タイトル箇所に年月を埋め込む
    // レコードの位置を統計情報設定ファイルから取得
    cellArea = props.getProperty("titile");
    // 統計情報設定ファイルに未設定のキーの場合はスキップ
    if (cellArea != null && cellArea.length() > 0 ){
        // 出力Cell位置を取得
        reference = new CellReference(cellArea);
```

```
            row = sheet.getRow(reference.getRow());
                cell = row.getCell(reference.getCol());
                // cellからフォーマットを取得
                String title = String.format(cell.getStringCellValue(), year, month);
                // 指定位置に値を出力
                cell.setCellValue(title);
        }

            // ファイル出力
            String outputFileName = String.format("統計情報_%d%02d.xlsx", year, month);
            Properties props2 = new Properties();
            props2.load(new FileInputStream("conf/apk_analysis.properties"));
            String outputPath = props2.getProperty("statistics.filepath") + "/" + outputFileName;
            log.debug("統計情報出力先[{}]", outputPath);
            try{
            FileOutputStream fos = new FileOutputStream(outputPath);
                wb.write(fos);
            } catch (IOException ioe) {
            log.error("統計情報出力エラー[{}]", ioe);
            System.exit(1);
            } finally {
                wb.close();
            }
```

```
            // 正常終了
            log.info("統計情報生成処理({}年{}月) 正常終了", year, month);
        } catch (Exception e) {
            log.error(e);
        } finally {
            // データベースクローズ
            if (dao != null) {
                dao.close();
            }
        }
    }
}
```

4.2 統計情報作成バッチシェル

統計情報作成バッチシェルのソースコードを以下に示す。

```
#!/bin/bash

LANG=ja_JP.UTF8
export LANG

cd /home/analyzer/bin;
```

```
if [ -d /var/data/request ]; then
        java  -Dlog4j.configurationFile="./conf/log4j2.xml" -jar apk_analysis.jar
fi
```

5 テンプレート・設定ファイル

5.1 返信メールテンプレートファイル

返信メールのテンプレートを以下に示す。

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <title>${analysis_response.requestNo}の解析結果 (JSSEC Android apk Analysis service) </title>
    <body font-style="MS Gothic" line-height="26px" margin-left="50px" margin-right="50px">
        <div align="left">
            【${analysis_response.applicationName}の解析結果】<br/><br/>

              この度は Android アプリケーション解析サービス をご利用頂きありがとうございました。本メールは、以下の解析依頼に関する静的解析の結果となります。<br/>
            <p style="margin-left: 20pt">
            解析受付日時：  ${analysis_response.requestYear}年  ${analysis_response.requestMonth}月  ${analysis_response.requestDay}日  ${analysis_response.requestHour}時  ${analysis_response.requestMinute}分<br/>
```

```
            受付番号：　${analysis_response.requestNo}<br/>
            解析対象アプリケーション名：${analysis_response.applicationName}<br/>
            </p>

            <p style="margin-left: 10pt">
                【解析結果】<br/>
                Androidアプリケーション静的解析を実施しました。<br/>
#if( ${analysis_response.result} == "00" )
                解析処理は成功しました。<br/>
#elseif( ${analysis_response.result} == "01" )
                想定時間内で解析できませんでした。解析できた内容のみ表示しています。<br/>
                検出無となっているものも解析が完了していれば、有であった可能性があります。<br/>
#elseif( ${analysis_response.result} == "02" )
                解析できませんでした。適切なファイルが指定されているかご確認ください。<br/>
#elseif( ${analysis_response.result} == "03" )
                解析できた内容のみ表示していますが利用者情報の送信判定が十分ではありません<br/>
                送信検出無となっているものも、有であった可能性があります。<br/>
#elseif( ${analysis_response.result} == "04" )
                大変申し訳ありませんが、システム不具合により解析できませんでした。<br/>
                <br/>
                時間をおいて再度お試しください。<br/>
#else
                大変申し訳ありませんが、システム不具合により解析できませんでした。<br/>
```

```
                <br/>
                時間をおいて再度お試しください。<br/>
#end

#if( ${analysis_response.result} == "00" || ${analysis_response.result} == "01" || ${analysis_response.result} == "03" )

            </p>
            <table width="95%" cellspacing="0" border="1px" bordercolor="#000000" border-spacing="5px">
                <tr bgcolor="#cccccc">
                    <td>項目名</td><td>解析結果内容</td>
                </tr>
                <tr>
                    <td>ハッシュ値（apk ファイルを一意に示す ID）</td><td>${analysis_response.apkHash}</td>
                </tr>
                <tr>
                    <td>apk のバージョン</td><td>${analysis_response.apkVersion}</td>
                </tr>
                <tr>
                    <td>apk ファイルの生成日付</td><td>${analysis_response.apkDate}</td>
                </tr>
                <tr>
                    <td>Permission</td><td>
    #if( ${analysis_response.permission.size()} >  0 )
```

```
                【検出したPermission】<br/>
        #foreach ($data in ${analysis_response.permission})
        ${data}<br/>
        #end
    #else
                Permissionは検出されませんでした。
    #end
                </td>
                </tr>
                <tr>
                <td>解析対象以外に利用者情報を送信する可能性について－動的な外部クラスの呼び出し有無</td><td>
    #if( ${analysis_response.classLoaderExecute.size()} >  0 )
                実行時に外部のクラスを呼び出し実行します。本解析結果に関わらず、呼び出したクラスから利用者情報を送信する可能性があ
ります。<br/>
                  【検出メソッド】<br/>
        #foreach ($data in ${analysis_response.classLoaderExecute})
        ${data}<br/>
        #end
    #else
                動的な外部クラスの呼び出しは検出されませんでした。
    #end
                </td>
                </tr>
```

```
                <tr>
                <td>解析対象以外に利用者情報を送信する可能性について－外部スクリプト・コマンドなどの呼び出し有無</td><td>
    #if( ${analysis_response.scriptExecute.size()} >  0 )
                外部スクリプト・コマンドなどを呼び出し実行します。本解析結果に関わらず、呼び出したスクリプト・コマンドなどから利用者情報を送信する可能性があります。<br/>
                【検出メソッド】<br/>
        #foreach ($data in ${analysis_response.scriptExecute})
        ${data}<br/>
        #end
    #else
                外部スクリプト・コマンドなどの呼び出しは検出されませんでした。
    #end
                </td>
                </tr>
            </table>
            <br/>
    #set( $EVALINFO_RESULT_CASE_ITEM = ["OS 生成 ID","端末固有 ID（IMEI)","契約者固有 ID(IMSI)","契約者固有 ID(ICCID)","デバイス固有 ID(MAC アドレス)","電話番号","メールアドレス","位置","アドレス帳","マイク","カメラ","加速度センサー","SMS"] )
            <p style="line-height: 10px">
            【利用者情報の送信】
            </p>
            <table  width="95%" cellspacing="0" border="1px" bordercolor="#000000" border-spacing="5px">
            <tr bgcolor="#cccccc">
```

```
            <td align="center">解析項目名</td><td align="center">送信検出</td>
            </tr>
    #foreach ($data in ${analysis_response.caseList} )
        #set( $counter = $velocityCount - 1 )
            <tr>
                <td align="center">$EVALINFO_RESULT_CASE_ITEM.get( $counter )</td>
        #if( ${data} == 1 )
                <td align="center" style="color: #00f">有</td>
        #elseif ( ${data} == 0 )
                <td align="center">無</td>
        #else
                <td align="center">―</td>
        #end
            </tr>
    #end
            </table>
            <p style="margin-left: 10pt">
                利用者情報の送信検出内容とプライバシーポリシーの記載内容を突合の上、整合性をご確認ください。整合性確認にあたって参考となる解析結果の突合手順書を含む、プライバシーポリシー調査シート、開発者向けのプライバシーポリシーの作成支援ツールなどの情報を JSSEC サイト（ http://www.jssec.org/jsas）にてダウンロードすることができます。<br/>
                <br/>
                ※本解析では、apk ファイル内の classes.dex を解析し、端末内の利用者情報を外部へ送信するロジックの有無を検出しております。classes.dex 以外は解析対象外ですが、それ以外に端末内の利用者情報を送信する可能性として、動的クラスローダーの利用有無、外部スク
```

```
リプトの呼び出す方法があり、それらの存在有無は検知しております。<br/>
                <br/>
#else
                <br/>
#end
                ※本解析サービスは、総務省の「スマートフォン上のアプリケーションにおける利用者情報の取り扱いに係る実証調査研究」に基づいています。この実証調査研究についての資料は JSSEC サイト（ http://www.jssec.org/jsas） で入手することができます。<br/>
                <br/>
                ※JSSEC として結果内容についての問い合わせは受け付けておりません。また、本解析結果について JSSEC では、いかなる損害やトラブルについても責任を負いませんのでご了承ください。本結果については上記実証調査研究の資料を基にご自身でご判断ください。<br/>
            </p>
        </div>
        <div style="line-height: 60pt" align="right">以上</div>
    </body>
</html>
```

5.2 統計情報設定ファイル

統計情報出力設定ファイルを以下に示す。

“B3” “F7” “F8” などの値は、テンプレートとなる Excel ファイルのセル位置を示している。

```
################################################################################
# 統計情報データと統計情報.xlsx の出力
```

```
################################################################################
# タイトル
titile=B3

# 解析申請総数
status_total=F7

# 解析完了総数
status_success=F8

#--------------------------------------
# 解析結果
#--------------------------------------
# 00:解析処理成功
status_00=F9
# 01：制限時間超過
status_01=F10

# 02：表層解析不可
status_02=F11

# 03:フロー解析エラー
status_03=F12
```

```
# 04:内部処理エラー
status_04=F13

#---------------------------------------
# アンケート：本システムの利用者
#---------------------------------------
# アプリ提供者（自らアプリを開発している）
service_user_1=F16

# アプリ提供者（他者に委託するなどで開発は自ら行っていない）
service_user_2=F17

# アプリ開発者（受託開発など自らの名では提供していない）
service_user_3=F18

# スマートフォンアプリに関する研究者
service_user_4=F19

# その他アプリ解析を必要とする事業者・個人
service_user_5=F20

# 予備１
```

```
service_user_6=F21

# 予備２
service_user_7=F22

# 予備３
service_user_8=F23

#-------------------------------------
# アンケート：本システムの利用回数
#-------------------------------------
# 初回
analysis_count_1=F26

# ２回目
analysis_count_2=F27

# ３回目
analysis_count_3=F28

# ４回目
analysis_count_4=F29
```

```
# ５回目以上
analysis_count_5=F30

#-------------------------------------
# アンケート：カテゴリー毎の件数
#-------------------------------------
# 一般
catetory_1=F33

#  Android Wear
family_1_1=F34

#  エンタメ
family_1_2=F35

#  カスタマイズ
family_1_3=F36

#  コミック
family_1_4=F37

#  ショッピング
family_1_5=F38
```

```
# スポーツ
family_1_6=F39

# ツール
family_1_7=F40

# ニュース＆雑誌
family_1_8=F41

# ソーシャルネットワーク
family_1_9=F42

# ビジネス
family_1_10=F43

# ファイナンス
family_1_11=F44

# メディア＆動画
family_1_12=F45

# ライフスタイル
```

```
family_1_13=F46

#  ライブラリ＆デモ
family_1_14=F47

#  医療
family_1_15=F48

#  音楽＆オーディオ
family_1_16=F49

#  教育
family_1_17=F50

#  健康＆フィットネス
family_1_18=F51

#  交通
family_1_19=F52

#  仕事効率化
family_1_20=F53
```

```
#  写真
family_1_21=F54

#  書籍＆文献
family_1_22=F55

#  通信
family_1_23=F56

#  天気
family_1_24=F57

#  旅行＆地域
family_1_25=F58

#  その他
family_1_26=F59

# ゲーム
catetory_2=F60

#  アーケード
family_2_1=F61
```

```
# アクション
family_2_2=F62

# アドベンチャー
family_2_3=F63

# カード
family_2_4=F64

# カジノ
family_2_5=F65

# カジュアル
family_2_6=F66

# シミュレーション
family_2_7=F67

# ストラテジー
family_2_8=F68

# スポーツ
```

```
family_2_9=F69

#  パズル
family_2_10=F70

#  ボード
family_2_11=F71

#  レース
family_2_12=F72

#  ロールプレイング
family_2_13=F73

#  音楽
family_2_14=F74

#  教育
family_2_15=F75

#  言葉
family_2_16=F76
```

```
#  雑学
family_2_17=F77

#  その他
family_2_18=F78

# ファミリー
catetory_3=F79

#  5 歳以下
family_3_1=F80

#  6～8 歳
family_3_2=F81

#  9 歳以上
family_3_3=F82

#  人気キャラクター
family_3_4=F83

#  アクション&アドベンチャー
family_3_5=F84
```

```
#  クリエイティビティ
family_3_6=F85

#  ごっこ遊び
family_3_7=F86

#  音楽＆動画
family_3_8=F87

#  教育
family_3_9=F88

#  頭の体操
family_3_10=F89

#  その他
family_3_11=F90

# 非公開（社内利用など）
catetory_4=F91

#  農業,林業,水産業,鉱業,建設業
```

```
family_4_1=F92

#  総合工事,建設業
family_4_2=F93

#  設備工事,製造業
family_4_3=F94

#  一般製造業,製造業
family_4_4=F95

#  印刷・出版,製造業
family_4_5=F96

#  その他の製造業,卸売・小売業,
family_4_6=F97

#  金融・保険業
family_4_7=F98

#  不動産業
family_4_8=F99
```

```
#  運輸・通信業
family_4_9=F100

#  電気・ガス・水道・熱供給業
family_4_10=F101

#  サービス業－放送業
family_4_11=F102

#  サービス業－広告／デザイン
family_4_12=F103

#  サービス業－法律／会計等
family_4_13=F104

#  サービス業－設計
family_4_14=F105

#  サービス業－医療
family_4_15=F106

#  サービス業－教育機関
family_4_16=F107
```

```
#  サービス業－各種団体
family_4_17=F108

#  サービス業－その他のサービス業
family_4_18=F109

#  公務
family_4_19=F110

#  利用業種は問わない
family_4_20=F111

#  その他(個人等)
family_4_21=F112

# その他
catetory_5=F113

# 解析結果
#  AndroidManifest.xml で定義された uses-permission 要素の総数
permission=F117
```

```
#  動的クラスローダーを実行しているアプリ件数
cle=F118

#  内部で Script を実行しているアプリ件数
sec=F119

#詳細解析結果（利用者情報の送信が検出されたアプリ件数）
# 利用者情報の送信
detail_result=F123

#  OS 生成 ID
os=F124

#  端末固有 ID（IMEI）
imei=F125

#  契約者固有 ID(IMSI)
imsi=F126

#  契約者固有 ID(ICCID)
iccid=F127

#  デバイス固有 ID(MAC アドレス)
```

```
macaddress=F128

#  電話番号
tel=F129

#  メールアドレス
mail=F130

#  位置
area=F131

#  アドレス帳
address_book=F132

#  マイク
microphone=F133

#  カメラ
camera=F134

#  加速度センサー
sensor=F135
```

```
#  SMS
sms=F136
```