iアプリ用オンライン RPG開発手法

HTTP通信とメモリ制御を中心に

双剣舞曲携帯サイト

iMenu → メニューリスト → ゲーム → ロールプレイング → 双剣舞曲オンライン

Contents

1. 概要
2. 技術紹介
    2.1 通信
    2.2 リソース管理
    2.3 メモリ管理
3. 終わり

©GAMES ARENA Co.,Ltd.

1. 概要

ゲーム概要

- タイトル「双剣舞曲オンライン」
- iアプリ用オンラインシミュレーションRPG
- クォータービュー形式
- 月額無料
- アイテム課金（アイテム、ガチャ、キャラ枠拡張、etc...）
- 対応機種：905i以上（SHのみ903iから）

各種仕様

ゲーム仕様

- 街中はMMO形式で、最大100キャラ+αを同時表示可能。
- 戦闘はMOのシミュレーション形式で、1戦闘に最大8人vs8人でプレイヤー同士の対戦可能（NPC含めると24vs24以上）
- キャラ作成時の顔選択や装備変更により、キャラクタの画像が変化するアバター形式

システム仕様

- SDカードを使用しない
- サーバ1台につき
  - 同時接続数：最大1500人
  - 戦闘のインスタンスエリア：最大900

iアプリ通信仕様

- HTTP通信のみ許容（starアプリはTCP/IP、UDP/IPも可）
- 毎回通信するたびにセッションを張りなおし（Keep-Alive不可）
- DL元のサーバとのみ通信可能（DXアプリ除く）
- HTTPに特定のパラメータを付与すると、docomo側のゲートウェイでユーザ毎のIDに変換されるため、ユーザ識別が出来る
- コネクションは同時には1本しか張れない
- 受信データが不完全な場合のチェックが必要

©GAMES ARENA Co.,Ltd.

環境

開発環境

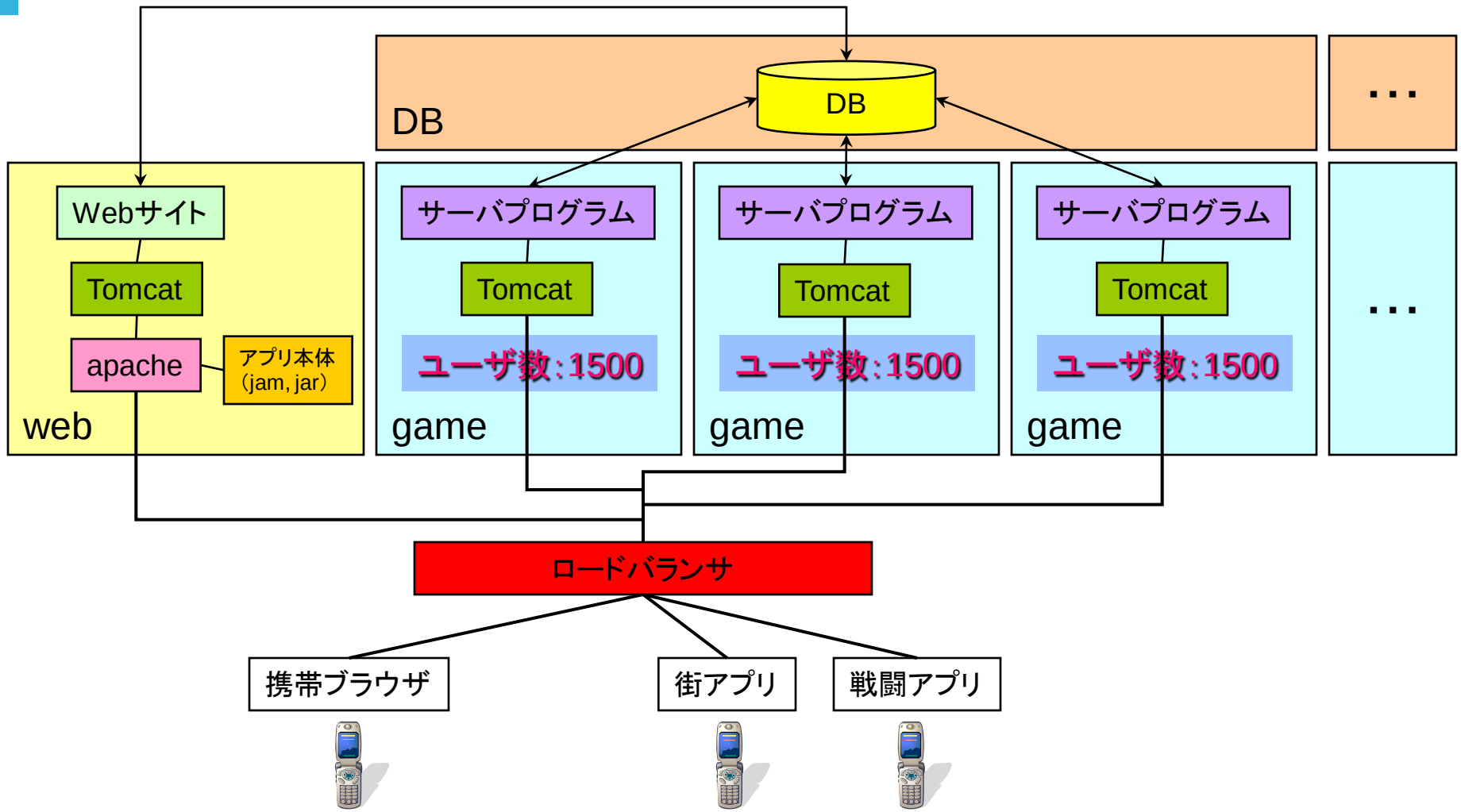
開発言語	Java（サーバ：JDK6、クライアント：CLDC1.1）
IDE	eclipse
ファイル管理	subversion
データ作成	Excel
ミドルウェア	apache2、Tomcat6、MySQL5
ツール類	Dojaエミュレータ、Cacti、proguard

どうしてC++でなくJavaで？

- メリット
  - アプリ・サーバでソースレベルでの処理の共有化が可能
    - 但し、iアプリ向けにCLDC1.1（jdk1.3相当）に合わせる必要がある（ジェネリック、オートボクシング、foreach文などが非対応、使えるクラスも少ない）
  - 言語レベルでのマルチスレッド対応（synchronized修飾子）
  - 各種ライブラリの充実（Logger、xUnit、concurrent、etc...）
  - Webとの親和性（Servlet、JSP）
    - データ閲覧・編集用のWeb管理ツールをJSPで簡単に作成可能
  - （PHPなどの軽量言語に比べて）高速動作
  - OS非依存で動作可能（Windows、Linux、...）

- デメリット
  - GC（Garbage Collection）により、メモリのコントロールが完全に制御できない

構成図


DB
DB
…
Webサイト
Tomcat
apache
アプリ本体
（jam, jar）
web
サーバプログラム
Tomcat
ユーザ数：1500
game
サーバプログラム
Tomcat
ユーザ数：1500
game
サーバプログラム
Tomcat
ユーザ数：1500
game
…
ロードバランサ
携帯ブラウザ
街アプリ
戦闘アプリ

2.1 通信

HTTP通信でオンラインゲーム

携帯電話とHTTP通信で作るオンラインゲーム

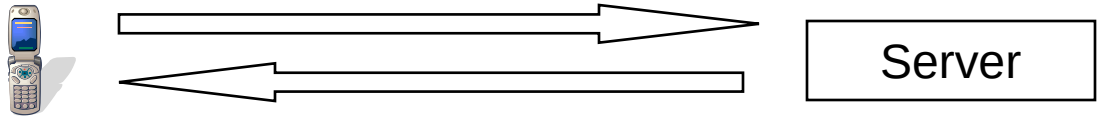
1. HTTPではサーバーからの通知が出来ない
   - 他人の行動タイミングをどうやって取得するのか
     - チャット発言のタイミング
     - 装備変更のタイミング
2. 通信ラグは2～4秒くらい当たり前
   - 処理の同期を取るのが大変
   - データ取得に時間のかかるUIの制御
   - iアプリではコネクションの接続に時間がかかる。
     - 毎回切断されてしまう仕様のため防ぐことができない

携帯電話オンラインゲーム最大の問題

- 通信が簡単に途切れること！
  - 地下鉄･トンネルによる一時的な圏外
  - 電話着信
  - 特定のキー操作
    - 以下のキーを押してしまうと通信切断
      - 電源キー
      - マルチキー
    - 操作ミスで発生（3キー押下時の電源キーなど）
    - 好きなタイミングで任意に発生させることができる

解決するには再送処理の実装が必要

- HTTP上に実装するには問題がある
  - 上りで切れたのか、下りで切れたのか分からない
    - 上りで切れた場合
      - サーバーに届いてないのでまだ処理されていない
    - 下りで切れた場合
      - サーバーでは既にクライアントからの要求を処理済み
  - サーバーに届いてもデータが切れている可能性
    - 不完全なデータは実行できないため、サーバーに届いていても実行されない可能性もある



適当な実装をしてしまうと・・・

- こんなことが発生してしまう！
  - 同じ処理を二重に実行してしまう
  - クライアントに結果が反映されないことが出てくる
- ゲームに与える影響は？
  - アイテムを二重に購入してしまう
  - チャット発言が届かない
  - HP0なのに攻撃してくる

オンラインゲームとして致命的！

重要ポイント

二重実行の発生しない安全な再送処理
を
「HTTP通信上に」構築する必要がある

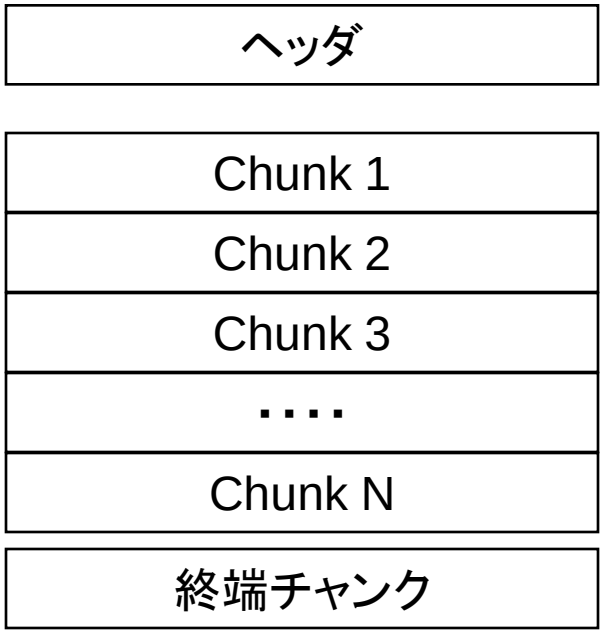
通信部分の実装

基本的な通信構造

- チャンク構造
  - 1通信で複数コマンドを発行することが出来る
  - オンラインゲームでは一般的？
  - チャンク　＝　1個のコマンド
    - ログイン要求
    - リソースの取得
    - チャット情報取得
    - 戦闘での行動
    - etc….
  - チャンクを複数個まとめたものを送受信する

詳細

- 構造
  - ヘッダ
    - 識別子
    - バージョン
    - シーケンスID
  - 実データ（チャンク個数分格納）
    - Chunk ID
    - サイズ
    - Chunk IDごとの固有データ
    - Chunkごとの終端マーク
  - 終端チャンク



クライアントの実装

- スレッド上で定期的に送信処理を行う
  - 送信したいチャンクをメモリ上のキューにどんどん追加
  - 一定のタイミングでキューに溜まったチャンクを送信する
- 結果を取得するまでは次の送信処理は行わない
  - これによりサーバーで実行したことを保証する
- クライアントでの再送処理（一般的アプリと同様）
  - 自動で一定回数のリトライ
  - それでも駄目ならダイアログ確認 → 無限リトライの抑制

サーバーの実装

- 基本的な流れ
  1. ヘッダの読み込み
     - クライアントの確認
     - バージョンの確認
  2. チャンクデータを読み込んで処理を実行
     - ログイン、チャット、攻撃実行 etc...
  3. クライアントへ結果を送信
     - ここもチャンク構造を使用して結果を返す

©GAMES ARENA Co.,Ltd.

再送処理を追加する

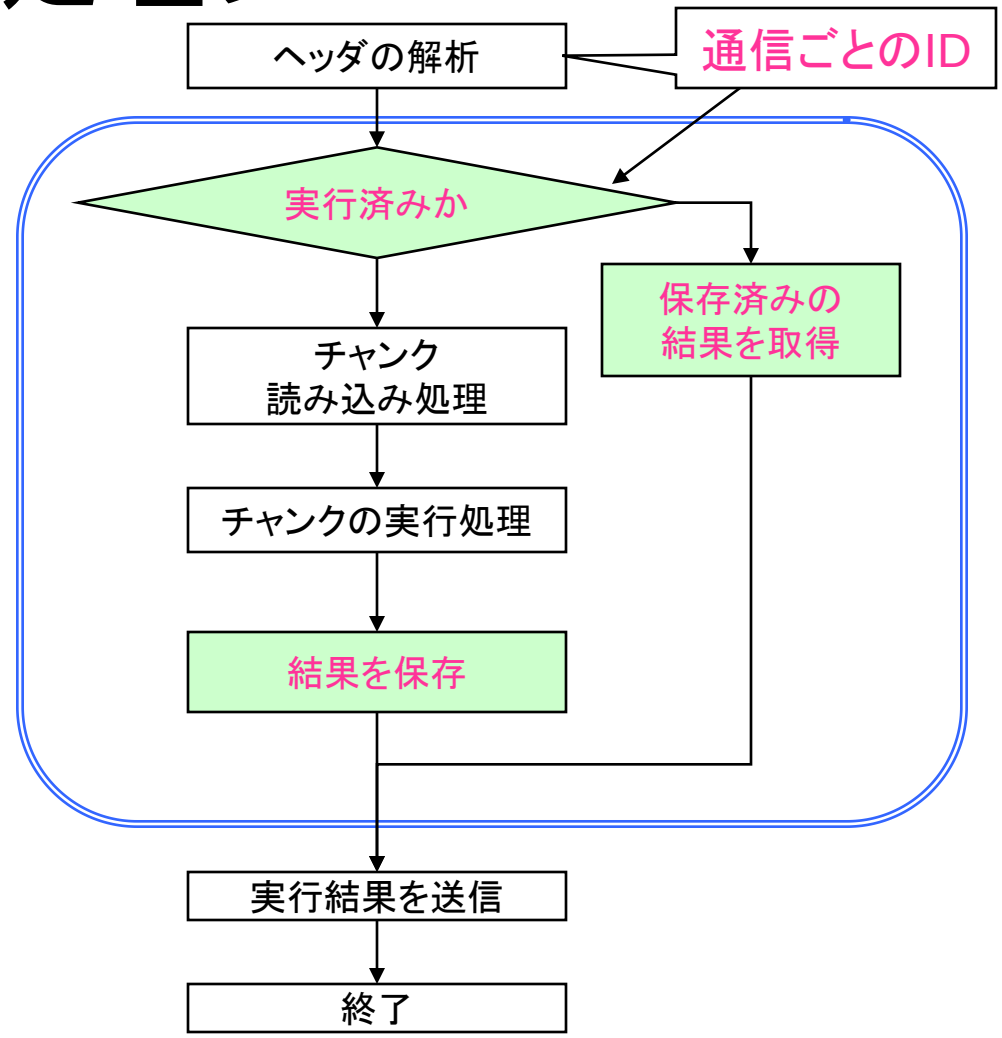
1. **データ構造が確認できるまで実行しない**
   - 途中でデータが切れている可能性があるため、最後までサーバーが受信できているのを確認してから処理の実行を始める必要がある
2. **再送データを二重実行せずに結果を返すようにする**
   - 再送と判断された通信は結果の送信のみ行う
   - どうすればいいのか
     - 実行した結果を保存しておく
     - 通信ごとに再送なのかどうかを判断し、未実行であれば処理の実行を行い、再送であれば結果の送信のみを行うようにする

©GAMES ARENA Co.,Ltd.

どうやって再送と判断するのか

- 通信1回ごとにIDを発行し判断材料とする
  - サーバーではユーザーと通信IDを対にしたキーを用いて実行結果をメモリ上に保存しておく
  - IDの発行の仕方には工夫が必要
    - かぶらないようにする必要がある
- 保存結果が格納されているかチェック
  - クライアント側は結果を受け取るまで同じデータを何回も再送するようになっているため、基本的には前回と同じIDかどうかを調べることで再送されたデータなのかを判定することが出来る

サーバー処理フロー


ヘッダの解析
通信ごとのID
実行済みか
保存済みの
結果を取得
チャンク
読み込み処理
チャンクの実行処理
結果を保存
実行結果を送信
終了

アプリ内での
リソース管理について

アプリ管理下のリソースサイズ

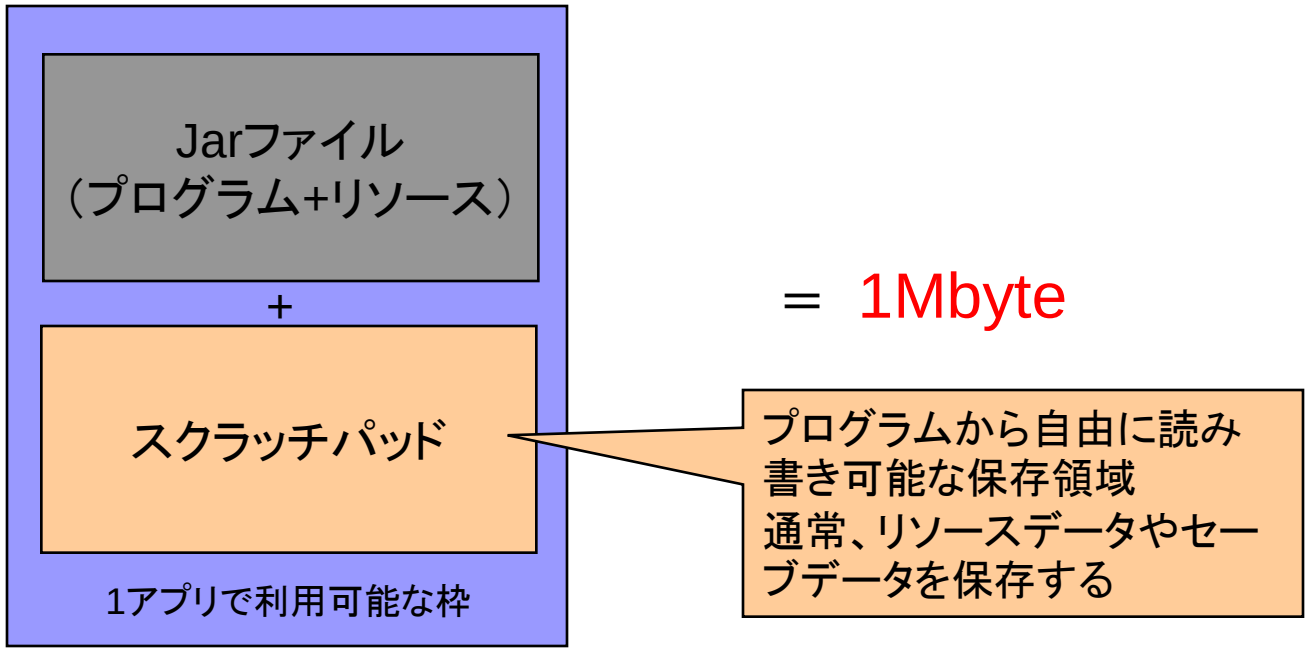
- 街アプリ
  - UI画像、ゲームデータ等：350kbyte
  - 1つの街で必要な、スクリプト+マップデータ等：350kbyte
  - キャラクタ画像：読み込めるだけ（最大100×1or2モーション）

- 戦闘アプリ
  - UI画像、ゲームデータ、スキル画像等：300kbyte
  - 1つの戦闘で用いる、キャラクタ画像（最大24×7モーション）+スクリプト+マップデータ等：100kbyte

©GAMES ARENA Co.,Ltd.

アプリで利用可能なストレージ

- スクラッチパッド
  - メガアプリの場合は1Mbyte（jarファイル+スクラッチパッド）利用可能
  - Jarファイル及びスクラッチパッドの領域設定は可変



アプリで利用可能なストレージ

- SDカード
  - 基本無料ゲームにおいてSD必須にするとユーザの登録数が減ってしまう恐れがあった
  - 前作の悠久の騎士団においてSDを使わないで作れたので、出来れば今回も使わないこととなった

↓

要求仕様としてSDカードは使用しないことに

- 端末内保存フォルダ
  - 保存フォルダをユーザが指定する必要がある
  - 保存時にシステムダイアログが出てしまいゲームの流れが途切れる
  - 必要のなくなったデータはユーザが手動で削除しない限りフォルダ内に残ってしまう

↓

ユーザにとって非常に煩雑で出来れば使用したくない

使用可能リソース容量

- 利用可能な容量について
  - メガアプリは1Mbyte（jarファイル+スクラッチパッド）利用可能
  - サービス継続中における機能拡張のことも考えて、街・戦闘アプリ各々、プログラムで500kbyteを確保

↓

スクラッチパッドに保存できるリソース容量は
各アプリたったの500kbyte！

アプリにおけるリソース取得と管理

- 結果
  - 全リソースを端末に保存することは不可能

↓

必要な時に適宜サーバから取得する！

- 実装方法
  - 街アプリ
    - UI画像、ゲームデータは起動時に毎回取得しメモリに展開
    - 1つの街で必要なスクリプト+マップデータはスクラッチパッドにキャッシュを行い、違う街へ行った際に取り直す
  - 戦闘アプリ
    - UI画像、ゲームデータ、スキル画像はスクラッチパッドに保存
    - 1つの戦闘で用いるキャラクタ画像+スクリプト+マップデータはアプリ起動時に毎回取得している

©GAMES ARENA Co.,Ltd.

ただし……

- キャラクタ画像は別問題
  - キャラクタ画像はフィールドにおいて適宜取得と破棄を繰り返す必要がある
  - 同時表示数は最大100以上
  - 装備画像は約15000ファイル以上

↓

スクラッチパッドやメモリには到底入らない

↓

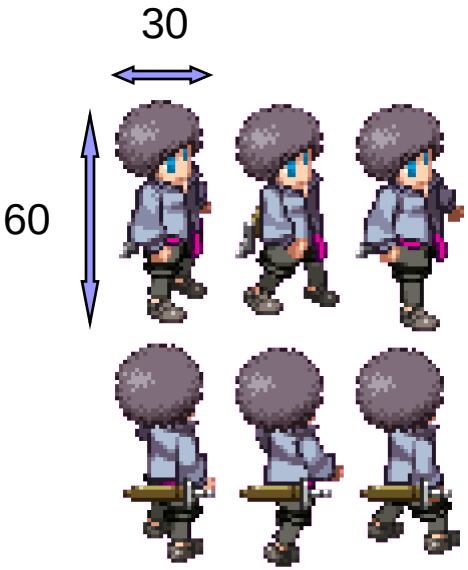
別途管理方法を考える必要がある

キャラクタ画像について

キャラクタ画像について

- 装備による画像変更箇所は以下の6箇所
  - 頭、胴、足、武器、顔、盾

- 1枚の画像は以下のような仕様
  - 1コマ：30x60（60x60）
  - アニメーション：3コマ×2パターン（上下）→ 6コマ
  - モーションパターン：複数存在（歩行、攻撃、座り etc…）

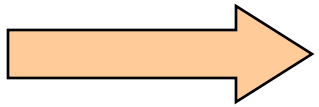


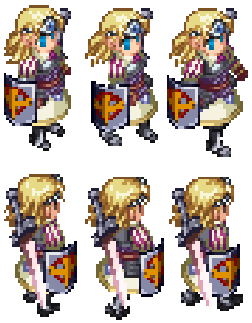
歩行時モーション

装備画像の組み合わせ

顔　　胴　　頭

足　　盾　　武器





キャラクタ画像
完成

各パーツ1枚当りの容量：2～4kbyte

アプリでのキャラクタ画像の扱いについて

～100体を実現するに当たって～

キャラクタ画像の取得

- キャラクタ画像の総計
  - 装備画像は約15000ファイル以上
    （更新によりサービス開始後もどんどん増えていく）
  - 敵キャラなどの画像も装備画像とは別途存在する

- キャラクタ画像の取得について
  - 同時表示100体を実現するに当たって、クライアントで画像を合成するとメモリもスクラッチパッドも足りない

↓

サーバから合成した画像を取得する

装備画像を個別で取得するよりも通信量が減る！

キャラクタ画像をサーバーで合成するメリットとデメリット

- メリット
  - 必要な分のみ適宜取得すればよいので装備画像の総容量は全く気にしなくて良い
  - ばらばらに装備画像をもらうと通信量が増えるが、合成済みの画像なら通信量が減る
  - 掲示板等WEB側のサービスでもサーバで合成したキャラクタ画像を使うことが出来る

- デメリット
  - アプリでのキャラクタの即時画像表示は不可能

フィールド上キャラクタの処理

- 同時キャラクタ表示数
  - キャラクタデータのみとしては100体くらいメモリ的には余裕
  - 画像も表示しようとすると33kbyte（歩行時モーション画像1枚のメモリ展開時）×100で3～4Mbyte必要

- メモリ使用量
  - 905以上の対応端末のメモリ総容量は13Mbyte～32Mbyteとまちまち
  - ネイティブヒープメモリ（画像を格納する専用メモリ）として使用可能な最低端末は4Mbyte

↓

端末の性能ごとにキャラクタ画像同時表示数を制御する必要あり
（キャラクタ同時表示数とキャラクタ画像同時表示数は別として考える）

オプション設定の姿勢

端末により描画能力、メモリ搭載量はまちまちであり、プログラム側で端末毎に1番最適な設定を自動で行うのは難しい。

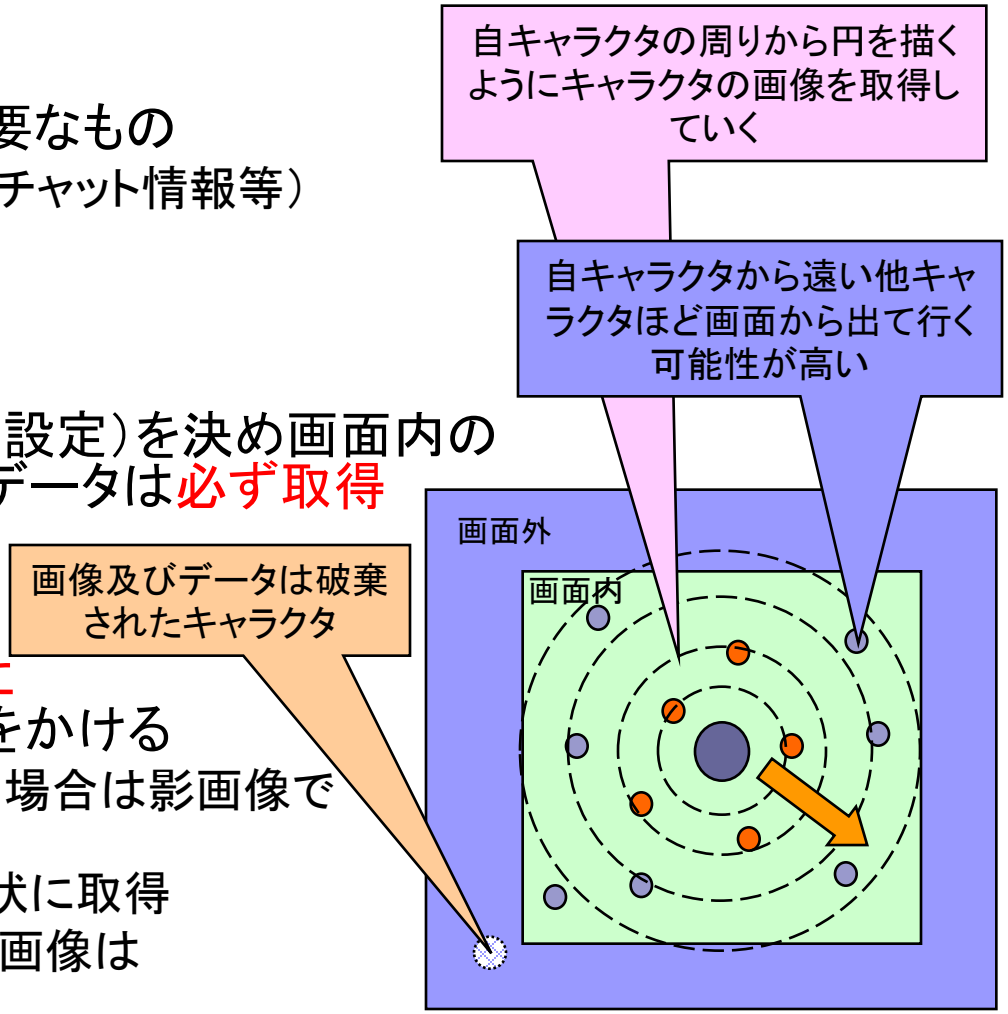
↓

PCゲームのようにコンフィグで設定できるようにし、ユーザが最も快適な状態を選んでもらうようにしてもよいのではないだろうか

- 実装したオプション項目
  - FPS、キャラクタ同時表示数、影表示、チャットの描画範囲等
  - キャラクタ画像取得数に関してはユーザが設定できるようにしてしまうとメモリ落ちがおきやすい設定を簡単に作れてしまうので項目には盛り込まなかった

フィールドにおける他キャラクタの表現について

- 他キャラクタを表現するのに必要なもの
  - キャラクタデータ（座標、名前、チャット情報等）
  - キャラクタ画像
- キャラクタデータ
  - 一定上限数（オプションでの設定）を決め画面内の一定範囲内の他キャラクタデータは必ず取得
- キャラクタ画像
  - 端末のメモリにより自動的にキャラクタ画像の取得制限をかける
    - キャラクタ画像が未取得の場合は影画像でキャラクタを表現
    - 自キャラクタの周りから円状に取得
    - 画面外に出たキャラクタの画像は即座に破棄



キャラクタデータとキャラクタ画像との取得タイミングの違い

キャラクタが画面に入ってきたときは影画像で表示

メモリ量さえ余裕があれば時間がたてばすべてのキャラクタの画像は表示される

サーバーメモリ関連

ガベージコレクション

- Java ＝ メモリ管理はGC
  - メモリの管理はJavaのVMに任せることになる
- JavaのGCについて
  - Mark and Sweep 方式
  - 世代別GC
  - GCの動作中は全てのJavaスレッドが停止
- 問題点
  - GCの処理時間はかなり長い
    - メモリの量に比例して長くなったりする
  - ゲームに関する定期動作が止まってしまう！

とりあえずそのまま実行してみた

- 条件
  - 負荷試験を実行
    - 1500人ログイン
    - 内 900人がソロで戦闘 → NPC 約40000体
  - NPCの数を多くして、AI、経路探索やチャットなどをどんどんする
- 結果
  - やっぱりGCが問題となってしまった
  - 1分に1回くらい
  - 3～5秒の全スレッド停止が発生
  - クライアントでどういう状況になるのか
    - 移動先決定→通信→移動処理(3～5秒）→通信→表示
    - 通信にも時間がかかるので、全体で10秒ほど応答無しの可能性

対応(1)
GCパラメータの調整

- 以下のオプションを使用してGC領域のチューニングを行う
  - -XX:NewRatio
  - -XX:SurvivorRatio
  - -XX:PermSize
  - -XX:MaxPermSize
- 容量設定の指針として、メモリの生存時間をリストアップする
  - 長い
    - アイテム、マップなどの基本ゲームデータ
    - ログイン中のキャラデータ
  - 中程度
    - 戦闘インスタンスで使用されるデータ
  - 短い
    - 通信時の処理に関わる部分
      - クライアントから受信したデータを実行するときの一時領域など
    - AIなどの経路探索

対応（1）
パラメータ調整結果

- GCの発生は避けられない
  - GCの発生間隔を長くすることは出来た
  - 長くなってもタイミングによって停止するのは変わらない
- 避けられないのはなぜか？
  - 一時オブジェクトが大量に発生してしまう
    - 1500人が数秒に1回接続
    - AI（経路探索）が、1秒に700体くらい動作
  - オブジェクトがどうしても生き残ってしまうため、OLD領域に移動し、GCの発生に繋がる

対応(2)
アルゴリズムの変更

- コンカレントGCを採用する
  - 設定方法
    - -XX:+UseConcMarkSweepGC
    - -XX:+CMSParallelRemarkEnabled
- コンカレントGCとは
  - GCに関する作業をスレッドを使用して裏で実行することにより、全スレッド停止の時間を短くしてくれる
    - 完全になくなるわけではない
  - そのかわり、全体的なスループットの低下が発生

対応(2)
アルゴリズム変更の結果

- メモリに関する問題は解決！
  - GCの停止時間を感じることは無くなった
  - スループットの低下は許容できる範囲
  - これを使用することによるプログラムの変更は無い。設定の変更のみで使用できる。

©GAMES ARENA Co.,Ltd.

Javaのメモリ調査用のツール

- JVM付属ツール
  - jstat
  - jconsole
- GCログ出力用のJVMオプション
  - verbose:gc
  - XX:+PrintGCTimeStamps
  - XX:+PrintGCDetails

©GAMES ARENA Co.,Ltd.

参考資料

- Java Performance Documentation
  - http://java.sun.com/docs/performance/index.html
- Java Tuning White Paper
  - http://java.sun.com/performance/reference/whitepapers/tuning.html
- (PDF) Memory Management in the Java HotSpot Virtual Machine
  - https://java.sun.com/j2se/reference/whitepapers/memorymanagement_whitepaper.pdf

ご清聴、ありがとうございました。

ご質問をお願いします。

連絡先:
上原 光晶 uehara@gamesarena.co.jp
町田 純専 machida@gamesarena.co.jp
鶴田 剛史 tsuruta@gamesarena.co.jp

©GAMES ARENA Co.,Ltd.