
Google™
09

Cache Pattern for Offline Web Applications

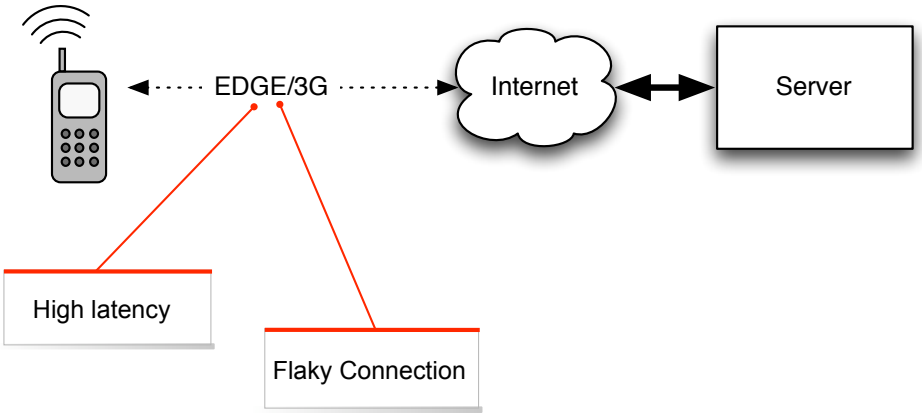
Robert Kroeger
June 27, 2009

Post your questions for this talk on Google Moderator:
code.google.com/events/io/questions

Why Mobile Web Applications?

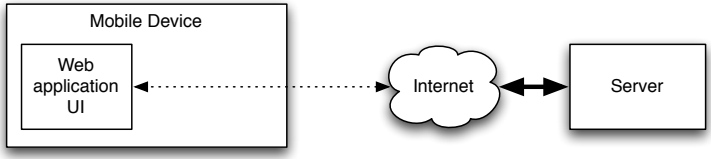
- **Pro**: server-side features that cannot be accommodated in a mobile device
  - Mobile devices have limited storage
  - Mobile devices have limited CPU
- **Pro**: ease of distribution
  - No app store approval process
  - Launch on your schedule
  - Update frequently
- **But**: use of wireless connections can make this much too slow

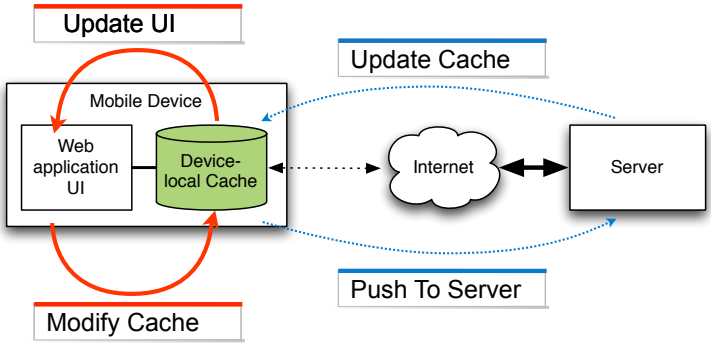
The Connection Problem



Caching Keeps Mobile Web Apps Responsive

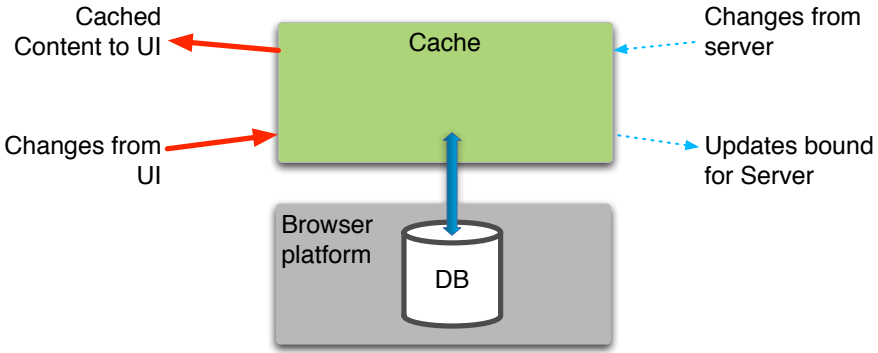
No *Synchronous* Connection From Web UI to Server





Implementing Cache Pattern

Storing the data

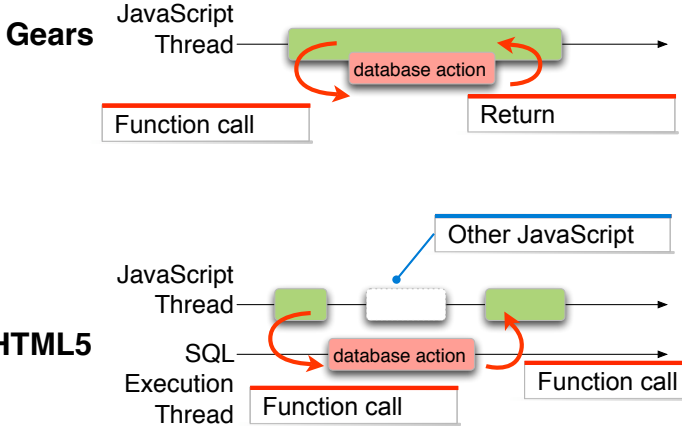


Cache Storage: Structured Storage

- Structured storage capability in HTML5 and Gears
  - A SQL database
  - Local to the client
  - Can access in absence of network connection
- Keep the cached contents here
  - Persistent across browser restarts
  - ACID properties maintain consistent database state

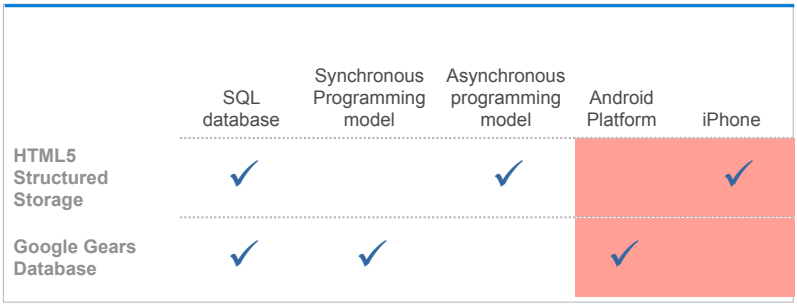
The Asynchronous Programming Model

Structured Storage: Gears != HTML5



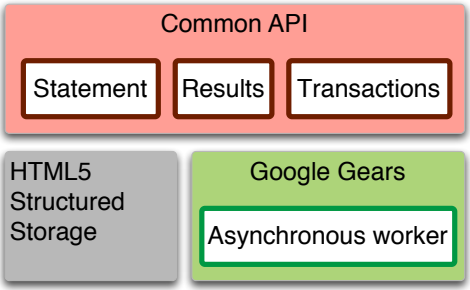
Portability?

Gears vs HTML5 Structured Storage



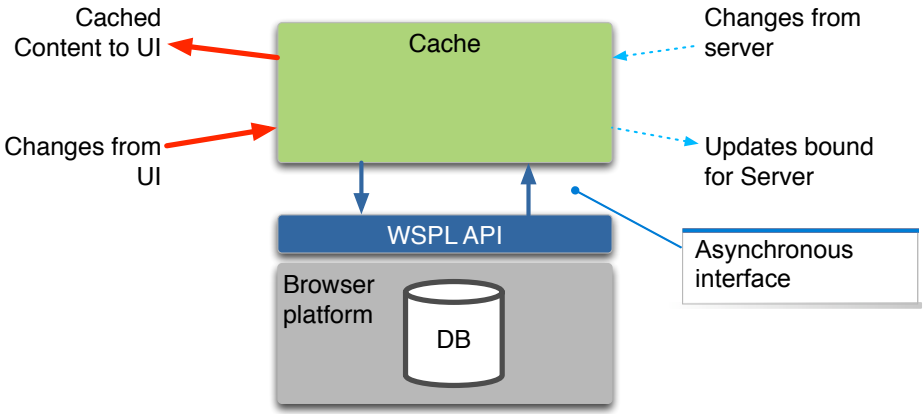
Web Storage Portability Layer (WSPL)

One API For Gears and HTML5

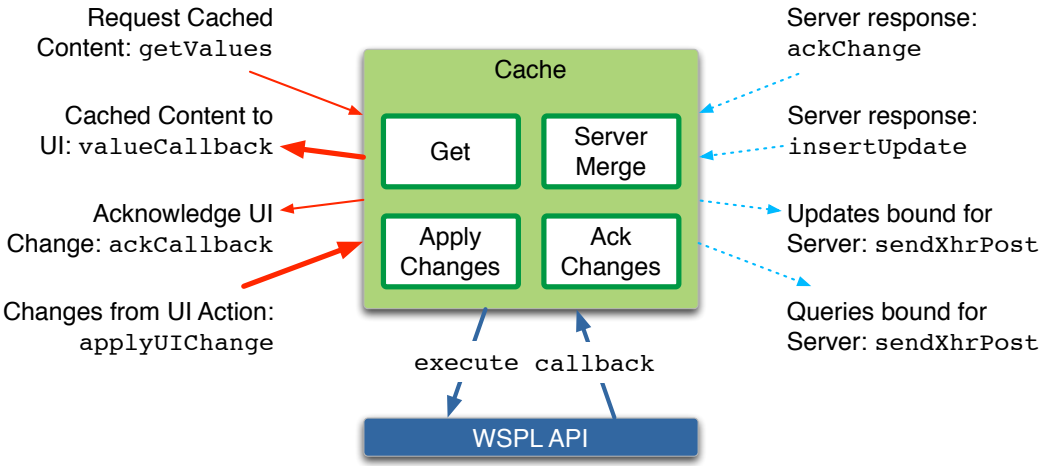


- Code to one common API
- Asynchronous programming model
- Gears worker implements the asynchronous programming model above synchronous database
- Available at code.google.com/webstorageportabilitylayer

Build Portable Cache above WSPL



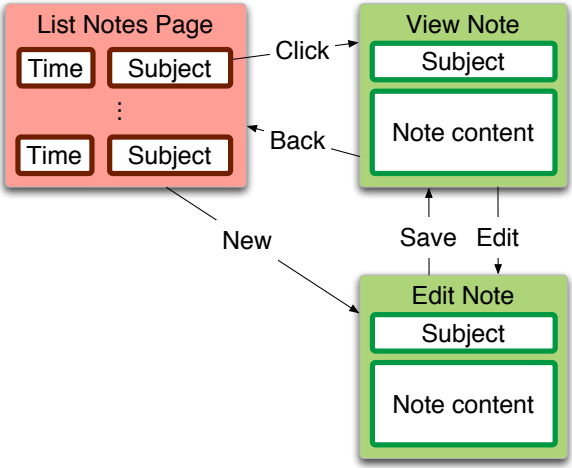
Asynchronous Cache Architecture



SimpleNotes

A Concrete Example of the Cache Pattern

- Toy application
  - keep notes
  - server side left as audience exercise
- Demonstrates use of the WSPL library
- Included with WSPL
- Three different “pages” in the application



Key Cache Design Aspects

- Hit Determination: what data is cached?
  - A contiguous range of notes
- Refresh: when to update the cached data with server changes?
  - Every request for a list of notes
- Eviction: what to discard to make room for new data
- Coherency: how to merge server changes into cache
  - Always send updates ahead of queries
  - Reapply actions for unacknowledged updates

Simple Notes Tables

```
google.wspl.simplenotes.Cache.CREATE_CACHED_NOTES_ =
      new google.wspl.Statement(
    'CREATE TABLE IF NOT EXISTS cached_notes (' +
        'noteKey INTEGER UNIQUE PRIMARY KEY,' +
        'subject TEXT,' +
        'body TEXT' +
    ');'
);

google.wspl.simplenotes.Cache.CREATE_WRITE_BUFFER_ =
      new google.wspl.Statement(
    'CREATE TABLE IF NOT EXISTS write_buffer (' +
        'sequence INTEGER UNIQUE PRIMARY KEY AUTOINCREMENT,' +
        'noteKey INTEGER,' +
        'status INTEGER,' +
        'subject TEXT,' +
        'body TEXT' +
    ');'
);

google.wspl.simplenotes.Cache.DETERMINE_MIN_KEY_ =
  new google.wspl.Statement(
  'SELECT MIN(noteKey) as minNoteKey FROM cached_notes;');
google.wspl.simplenotes.Cache.DETERMINE_MAX_KEY_ =
  new google.wspl.Statement(
  'SELECT MAX(noteKey) as maxNoteKey FROM cached_notes;');
```

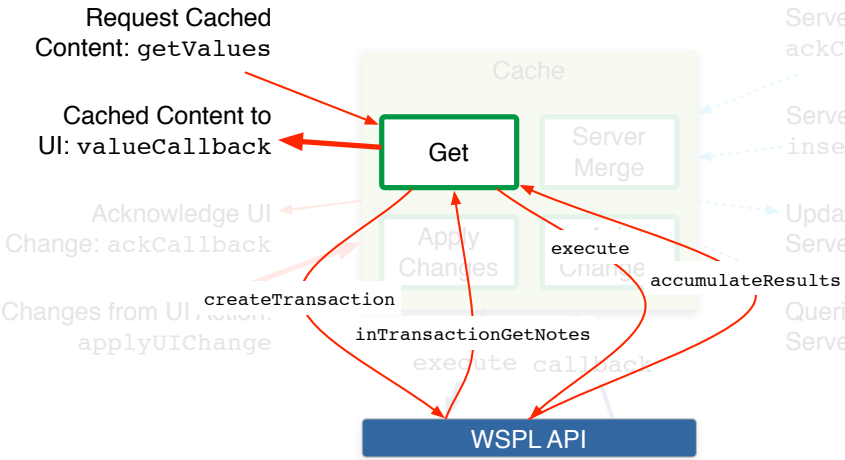
Separate write buffer: actions vs state

Simple Notes Creation

```
google.wspl.simplenotes.Cache.prototype.startCache = function(callback) {
  var statc  = 0; var self = this;

  var perStatCallback = function(tx, result) {
    google.logger('perStatCallback');
    if (statc == 4) {
      self.start_ = (result.isValidRow()) ? result.getRow().minNoteKey : -1;
      self.serverStart_ = self.start_; // Temporary. Remove when server exists.
    } else if (statc == 5) {
      self.end_ = (result.isValidRow()) ? result.getRow().maxNoteKey : -1;
      self.serverEnd_ = self.end_; // Temporary. Remove when server exists.
    }
    statc++;
  };

  this.dbms_.executeAll([
      google.wspl.simplenotes.Cache.CREATE_CACHED_NOTES_,
      google.wspl.simplenotes.Cache.CREATE_WRITE_BUFFER_,
      google.wspl.simplenotes.Cache.CREATE_UPDATE_TRIGGER_,
      google.wspl.simplenotes.Cache.CREATE_REPLAY_TRIGGER_,
      google.wspl.simplenotes.Cache.DETERMINE_MIN_KEY_,
      google.wspl.simplenotes.Cache.DETERMINE_MAX_KEY_],
      {onSuccess: perStatCallback, onFailure: this.logError_},
      {onSuccess: callback, onFailure: this.logError_});
  google.logger('finished startCache');
};
```

Asynchronous here too

`getValues` Hit Call Flow



`getValues` Example Code (1)

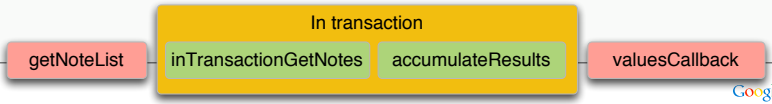
```
google.wspl.simplenotes.Cache.prototype.getValues = function(type,
    query, valuesCallback) {

  // Reduce any query to what would be available from the server
  query[0] = Math.max(this.serverStart_, query[0]);
  query[1] = Math.min(this.serverEnd_, query[1]);

  if (type == 'list') {
    this.getNoteList_(query[0], query[1], valuesCallback);
  } else if (type == 'fullnote') {
    this.getOneNote_(query[0], valuesCallback);
  }
};
```

Same idea as `getNoteList_`

getValues Example Code (2)

```
google.wspl.simplenotes.Cache.prototype.getNoteList_ = function(start, end,
    valuesCallback) {
  var notes = [];

  var accumulateResults = function(tx, result) {
    for(; result.isValidRow(); result.next()) { notes.push(result.getRow()); }
  };

  var inTransactionGetNotes = function(tx) {
    tx.execute(google.wspl.simplenotes.Cache.LIST_CACHED_NOTES_.
        createStatement([start, end]), {
        onSuccess: accumulateResults,
        onFailure: this.logError_});
  };

  var hit = this.isCacheHit_(start, end);
  this.dbms_.createTransaction(inTransactionGetNotes, {onSuccess: function() {
    valuesCallback(notes, hit);
  }, onFailure: this.logError_});

  if (hit) {
    this.fetchFromServer(this.start_, this.end_); // Refresh
  } else {
    this.fetchFromServer(Math.min(this.start_, start), Math.max(this.end_, end));
    this.lastMiss_ = {callback: valuesCallback, start: start, end: end};
  }
};
```

Need a cache membership scheme



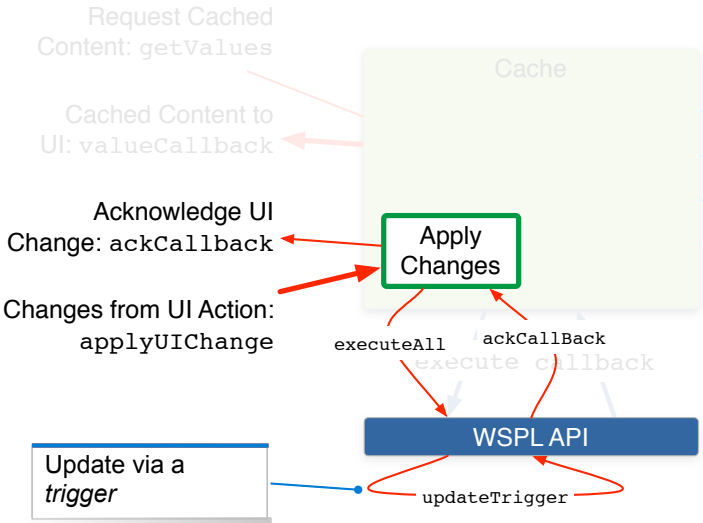
Hitting and Missing

- isCacheHit
  - Application Specific
  - Simple approach here: cache contains a contiguous range of notes.

```
google.wspl.simplenotes.Cache.prototype.isCacheHit_ = function(start, end) {
  return start >= this.start_ && end <= this.end_;
};
```

  - Cache maintains the start and end values in JavaScript
  - Cache grows its range by making a server request for the missing values.
- Avoid database accesses

`applyUIChange` Call Flow

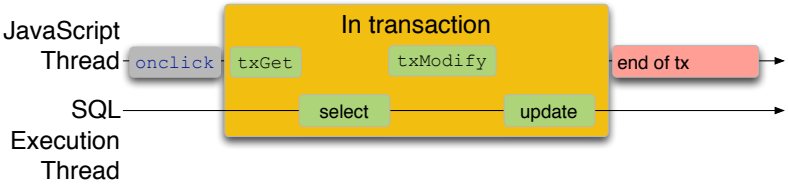


applyUIChange Code

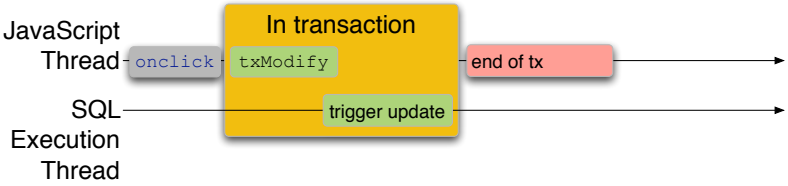
```
google.wspl.simplenotes.Cache.INSERT_UI_UPDATE_ =
    new google.wpsl.Statement(
        'INSERT INTO write_buffer (' +
          'noteKey, status, subject, body )' +
        'VALUES(?,?,?,?);');

google.wspl.simplenotes.Cache.prototype.applyUiChange = function(noteKey,
    subject, body, ackCallback) {
  var self = this;
  var update = [noteKey, 2 | 8, subject, body];
  var stat = google.wspl.simplenotes.Cache.INSERT_UI_UPDATE_.createStatement(
      update);

  this.dbms_.execute(stat, null, {onSuccess: function() {
    ackCallback(noteKey);
  }, onFailure: function (error) {
    self.logError_(error);
    ackCallback(-1);
  }});
};
```

Why Triggers: Avoid Ping-Ponging

Long Transactions Are Bad



- Read-modifiy-write operations are inefficient
- A trigger runs entirely inside SQL thread



`applyUIChange` Trigger Code

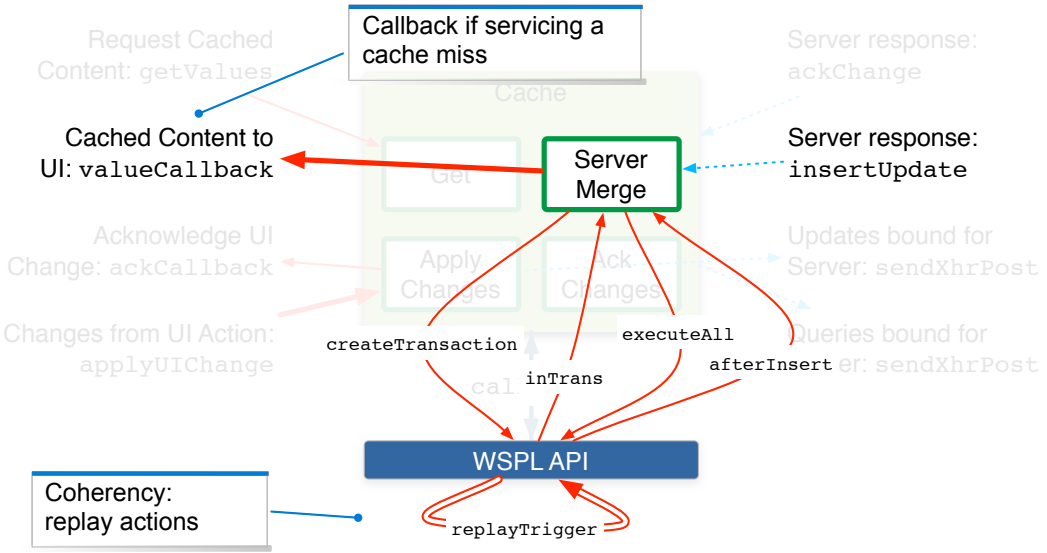
```
google.wspl.simplenotes.Cache.CREATE_UPDATE_TRIGGER_ =
    new google.wspl.Statement(
        'CREATE TRIGGER IF NOT EXISTS updateTrigger ' +
        'AFTER INSERT ON write_buffer ' +
        'BEGIN ' +
        '  REPLACE INTO cached_notes ' +
        '    SELECT noteKey, subject, body ' +
        '      FROM write_buffer WHERE status & 8 = 8; ' +
        '  UPDATE write_buffer SET status = status & ~ 8; ' +
        'END;'
    );
```

Update cache from `write_buffer`

Trigger is only way to execute several statements

- `status`: a bit-field of states
  - 1 The update is inflight to the server
  - 2 The update needs to be (re)sent to the server
  - 8 The update needs to replayed against the cache

`insertUpdate` Call Flow



insertUpdate Code

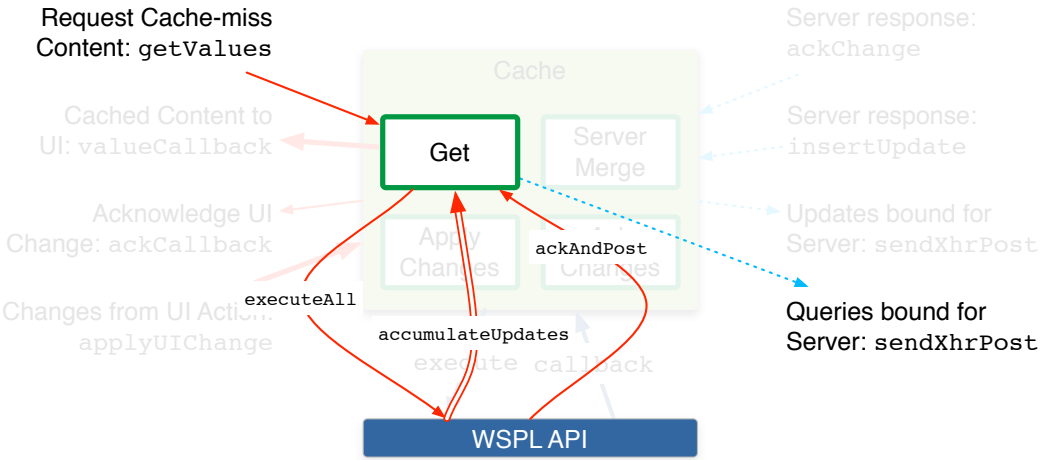
```
google.wspl.simplenotes.Cache.prototype.insertUpdate = function(notes) {
  var self = this; var stats = [];
  var start = notes[0].noteKey; var end = notes[0].noteKey;

  for (var i = 0; i < notes.length; i++) {
    stats.push(google.wspl.simplenotes.Cache.INSERT_NOTE_.
        createStatement([notes[i].noteKey, notes[i].subject, notes[i].body]));
    start = Math.min(start, notes[0].noteKey);
    end = Math.max(end, notes[0].noteKey);
  }
  stats.push(google.wspl.simplenotes.Cache.EVICT_.createStatement([start, end]));
  stats.push(google.wspl.simplenotes.Cache.FORCE_REPLAY_);

  var inTrans = function(tx) {
    self.start_ = start; self.end_ = end;
    tx.executeAll(stats);
  };

  var afterInsert = function(tx) {
    if (this.lastMiss_ &&
        this.isCacheHit_(this.lastMiss_.start, this.lastMiss_.end)) {
      this.lastMiss_.callback(notes);
      this.lastMiss_ = undefined;
    }
  };
  this.dbms_.createTransaction(inTrans, {onSuccess: afterInsert,
      onError: this.logError_});
};
```

Must update here to align JS change with transaction boundary

Callback if this update from the server satisfies a pending request

`fetchFromServer` Call Flow

Handling Cache Misses and Refresh

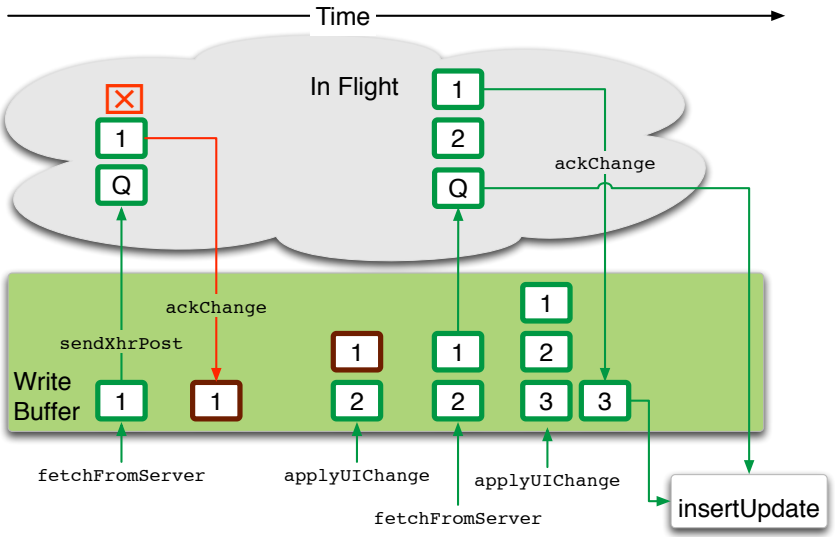


fetchFromServer Code

```
google.wspl.simplenotes.Cache.prototype.fetchFromServer = function(start,
    end) {
  var now = this.dbms_.getCurrentTime();
  if (start >= this.start_ && end <= this.end_ && now - this.lastRefresh_ <
      google.wspl.simplenotes.Cache.TIME_BETWEEN_REFRESH_) {
    return;
  }

  var updates = []; var self = this; var flag = 1; var sql = [];
  sql.push(google.wspl.simplenotes.Cache.GET_UPDATES_TO_RESEND_);
  sql.push(google.wspl.simplenotes.Cache.MARK_AS_INFLIGHT_);

  var accumulateUpdates = function(tx, rs) {
    if (flag == 1) {
      for(; rs.isValidRow(); rs.next()) { updates.push(['u', rs.getRow()]); }
      flag++;
    }
  };

  var ackAndPost = function() {
    updates.push(['q', {start: start, end: end}]);
    self.sendXhrPost(updates);
  };

  this.dbms_.executeAll(sql,
      {onSuccess: accumulateUpdates, onFailure: this.logError_},
      {onSuccess: ackAndPost, onFailure: this.logError_});
};
```

Maintaining Server Consistency

Resend failures before queries, then reapply actions



Miscellaneous Tidbits

- Messages to server
  - JSON-encoded
  - via XHR POST
- See the WSPL open source distribution for the remaining missing bits
- Asynchronous model decouples the execution order from the display order
- Triggers crucial to rapid development
- SimpleNotes glosses over some hard problems:
  - Database too slow to serve as application model
  - Too much data traffic

Summary

- Cache pattern workable in real products: Mobile Gmail for iPhone and Android.
- WSDL library same
- Workable local storage solution across iPhone and Android

Q & A

Post your questions for this talk on Google Moderator:
code.google.com/events/io/questions


Google™
09