ウェブ関連最新技術動向

インターシステムズジャパン株式会社

InterSystems®
Health | Business | Government

Webクライアント開発アプローチ

標準Web技術

ネイティブ

ハイブリッド

ウェブアプローチ

- HTML 5とCSS3を利用

Java Scriptフレームワーク

フレームワーク概要

DOM探索と操作

イベント操作

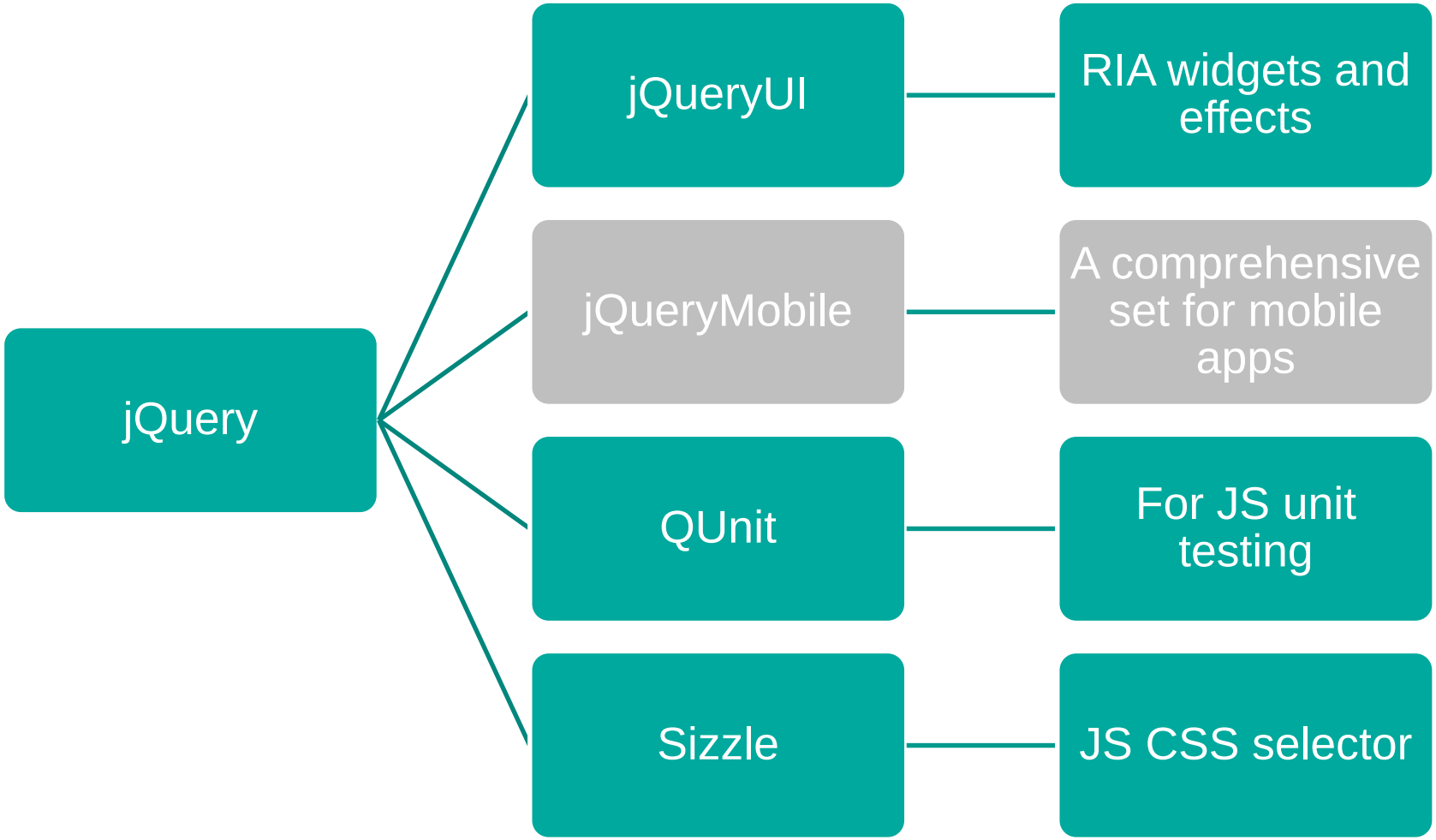
Ajax呼出し

ウィジェット

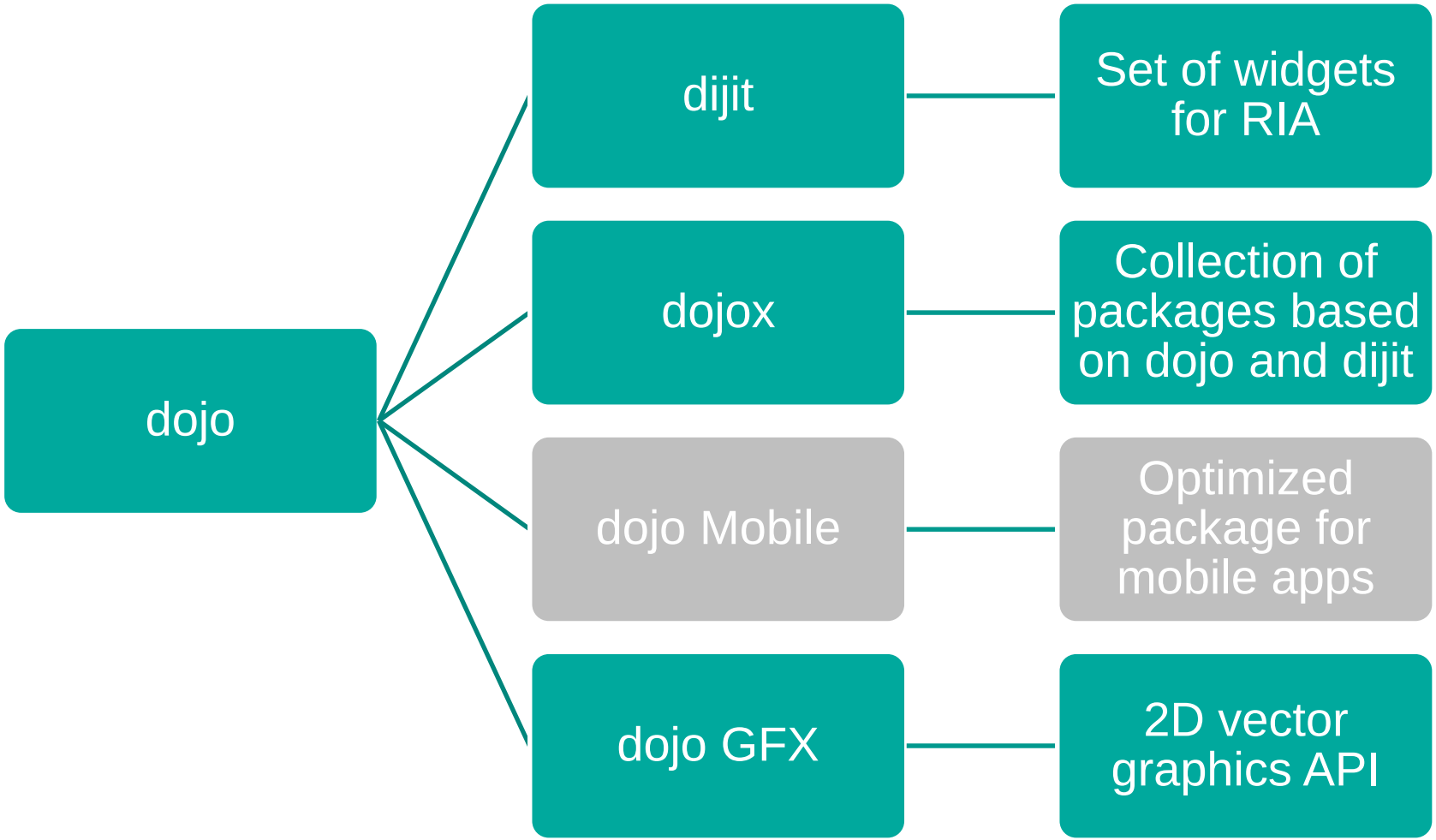
アニメーション

モバイルフレームワーク

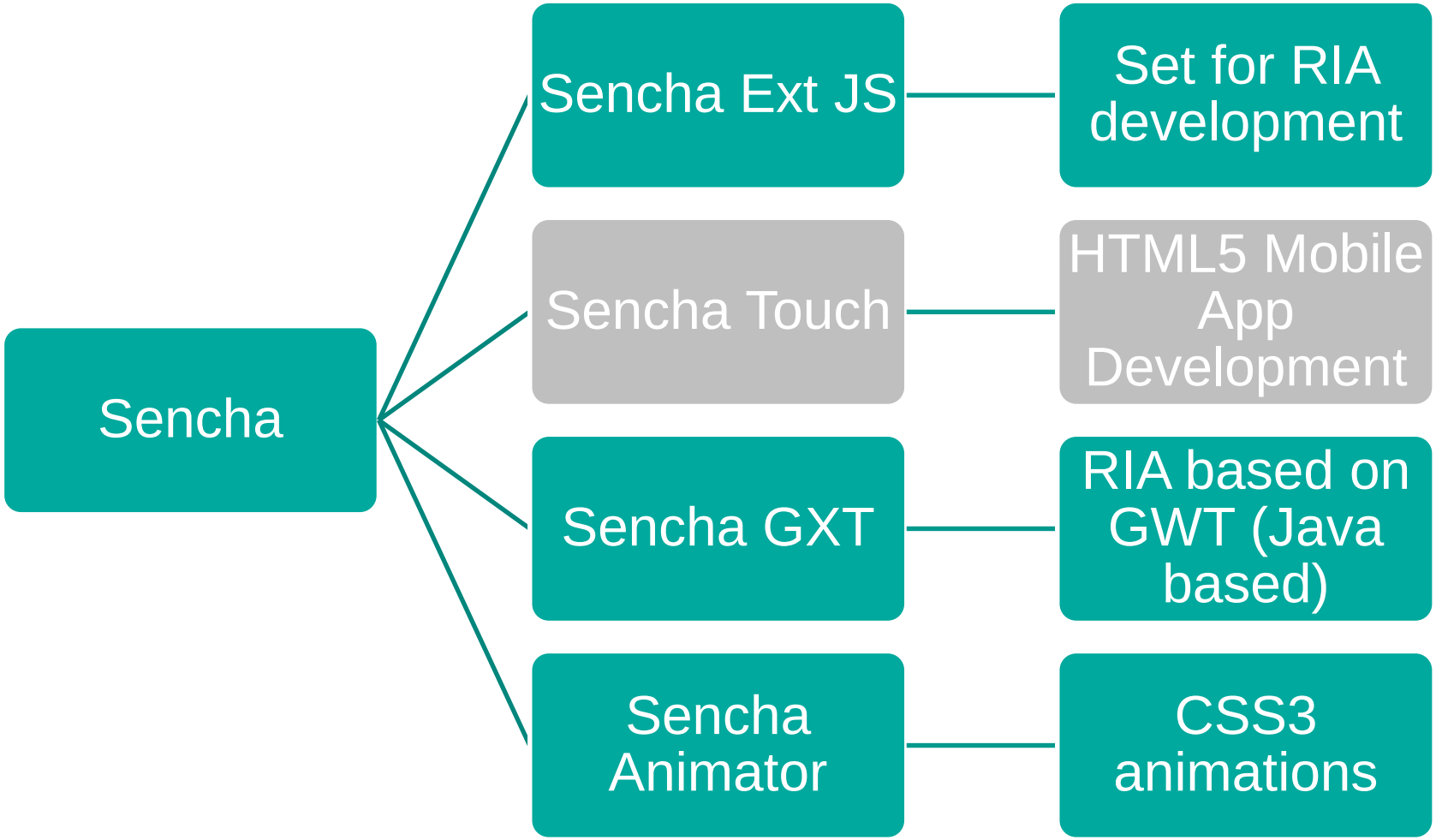
jQuery



dojo



Sencha



フレームワークはどれも似たり寄ったり

モバイル専用のパッケージ

そのほかのパッケージも必要

簡潔なHTMLを記述

DOMが準備された後で変換

開始コード

Default HTML

jQuery sample

```
1 window.onload = function() {
2 alert( "welcome" );
3 }
```

```
1 $( document ).ready(function() {
2 // Your code here
3 });
```

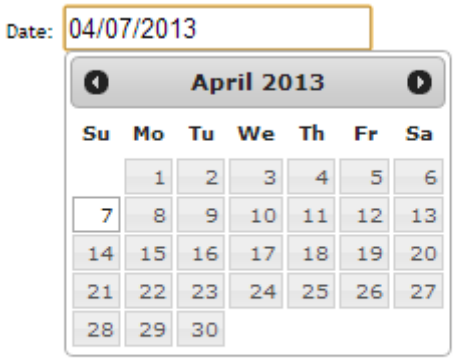

ウィジェットはどのように動作する?

ID付HTMLエレメント

エレメントのjQuery関数を呼び出す

```
<p>Date: <input type="text" id="datepicker" /></p>

<script>
$(function() {
$( "#datepicker" ).datepicker();
});
</script>
```



さらなる小さなマジック

```
<div id="clickThisDiv">Click Me!</div>

<div id="myDiv"
        style="background-color:red;width:200px;height:200px;position:
relative; left: 10px;"></div>
```

```
$('#clickThisDiv').click(function() {
animateDiv();
});

function animateDiv() {
$('#myDiv').animate({
left : '+=50',
height: 'toggle'
}, 1000);
};
```


モバイルについてはどうか?

宣言的アプローチを使用（テンプレート）

詳しくは…

querymobile.com/1.3.0/docs/widgets/panels/

データはどこに?

サーバーへAJAX呼出しを使う

- 動的DOM注入
- データ取得
- スクリプトロード

jQuery: $.getJSON(url,data,callback)
サーバー側: JSON とCSP/ZEN/Zen Mojo/REST

HTML5 + CSS3 vs. JS Frameworks

大きさとロード時間

アクセス性

互換性

 |

困難な点

フレームワークを学習しなければならない

異なるランタイムの異なる呼出しで同じ結果が得られる

ウェブアプローチ

良い点	悪い点
１つのコードベースだけ必要	デバイス機能の限られたアクセス
いつでも起動、修正が可能	支払プロセスがない
インストレーションプロセス不要	複数ブラウザのサポート
既存のウェブアプリケーションを化粧直しできる	

ネイティブアプローチ

iOS → Objective-C、Swift

Android → Java

Windows → .NET

ネイティブアプローチ

良い点	悪い点
デバイス機能への完全アクセス	ソフトウェア更新をスキップできる
簡単な支払プロセス	開発費高価
美しい見栄え	複数のコードベース必要
モバイルアプリケーションより高速に実行	

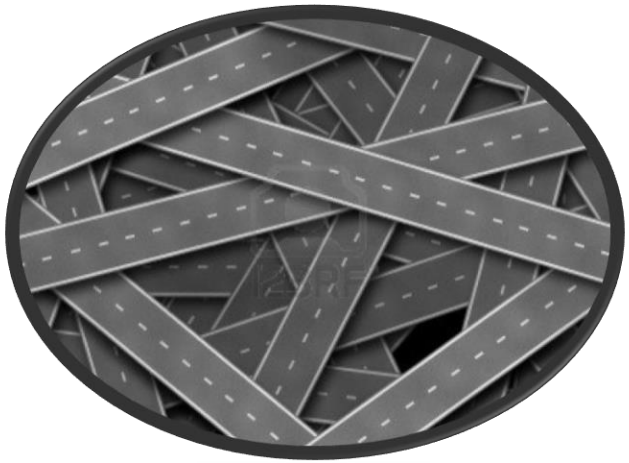
たくさんの道のり…

ネイティブ開発

- より長いリリースサイクル
- デバイスタイプ毎にコードの書き換え

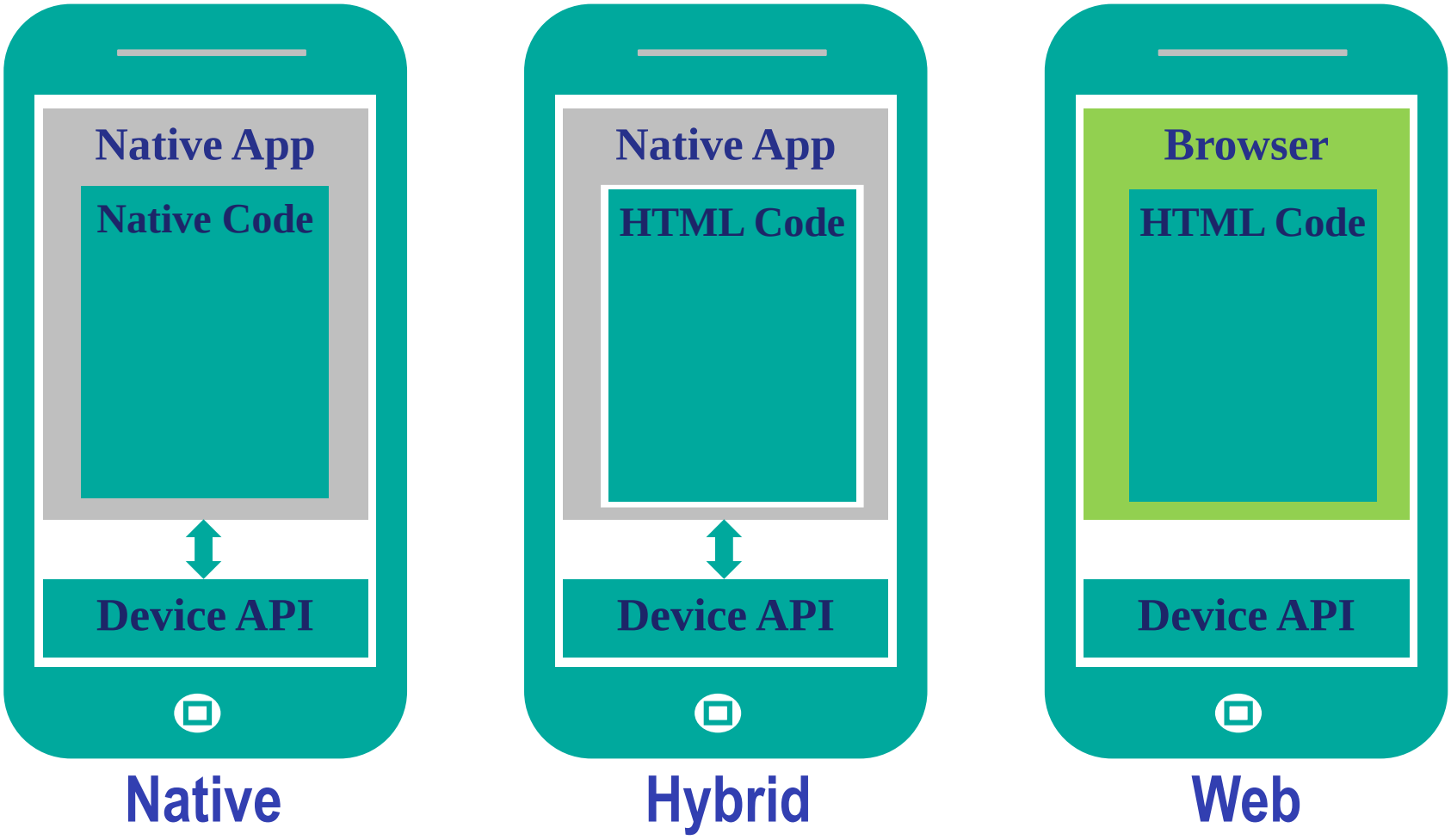
ウェブベース開発

- ネイティブアプリではない
- ブラウザの感触
- デバイス機能への限られたアクセス



 |

PhoneGapはハイブリッド



 |

PhoneGap

ハイブリッド開発には、PhoneGapを推奨

ステートフル　VS　ステートレス

モバイルの特性

- 消費電力
- 無線LANの不安定性
- 通信の帯域
- スケーラビリティ

ステートレスな通信が望ましい

HTTP　+　JSON

 |

JSON vs. XML

JSON	XML
データ構造	データ構造
検証の仕組みなし	XSD
名前空間（ネームスペース）なし	名前空間（ネームスペース）あり (複数使用可)
解析は高速、特にJavascript eval()を使うと	解析にはXpathなどを使ったXMLドキュメント解析が必要

Zenのおさらい

約10年前にリリース

製品として完全にサポート保証

マルチページアーキテクチャー

サーバー側コンポーネントモデル

- 全ての必要なHTMLとJavaScriptコードは、サーバーからクライアントに送られる
- オンラインの接続と十分な帯域が必要

イントラネットアプリケーションにはそれで良かった

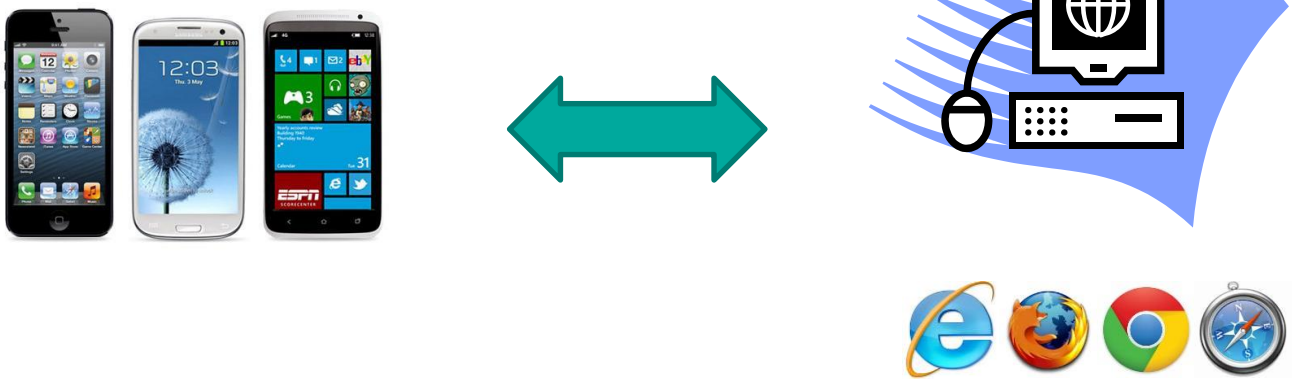
昨今ユーザーの期待は

- 低い帯域でも良い応答
- オフラインモード

Zen Mojo

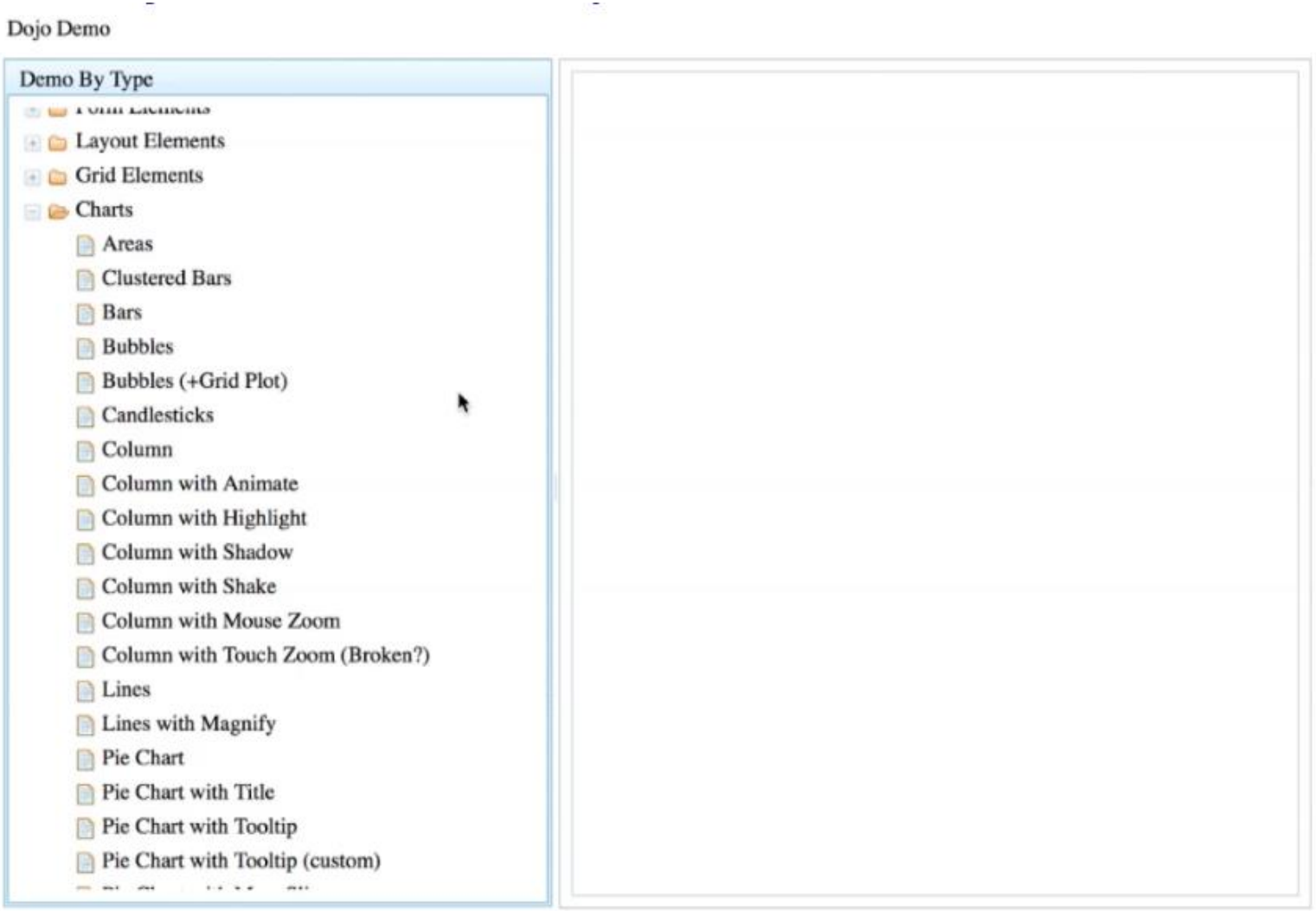
異なるプラットフォーム用の１つのフレームワークであり、１つの開発パラダイム

- デスクトップアプリケーション
- モバイルアプリケーション



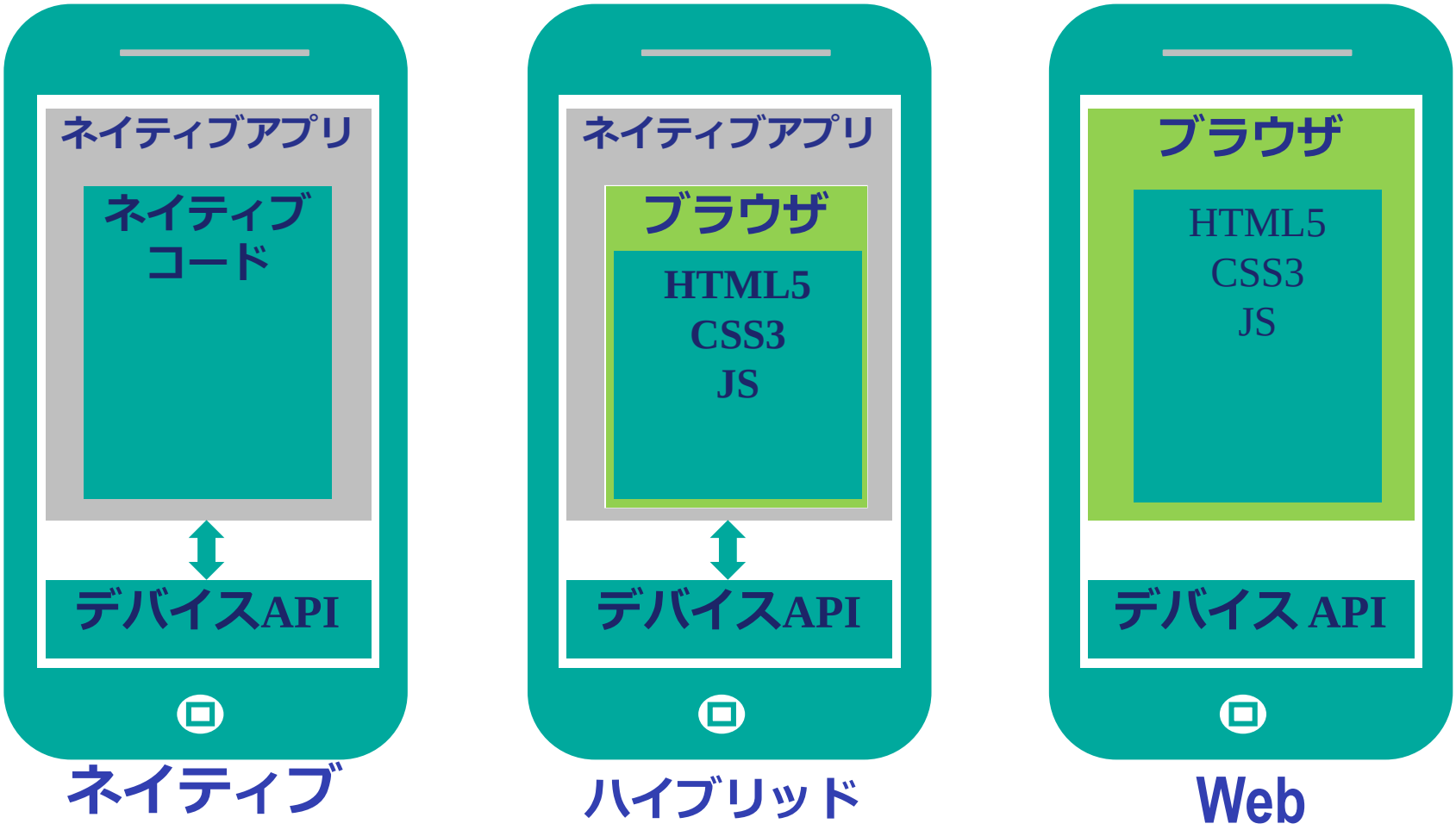
 |

デスクトップ用Zen Mojo

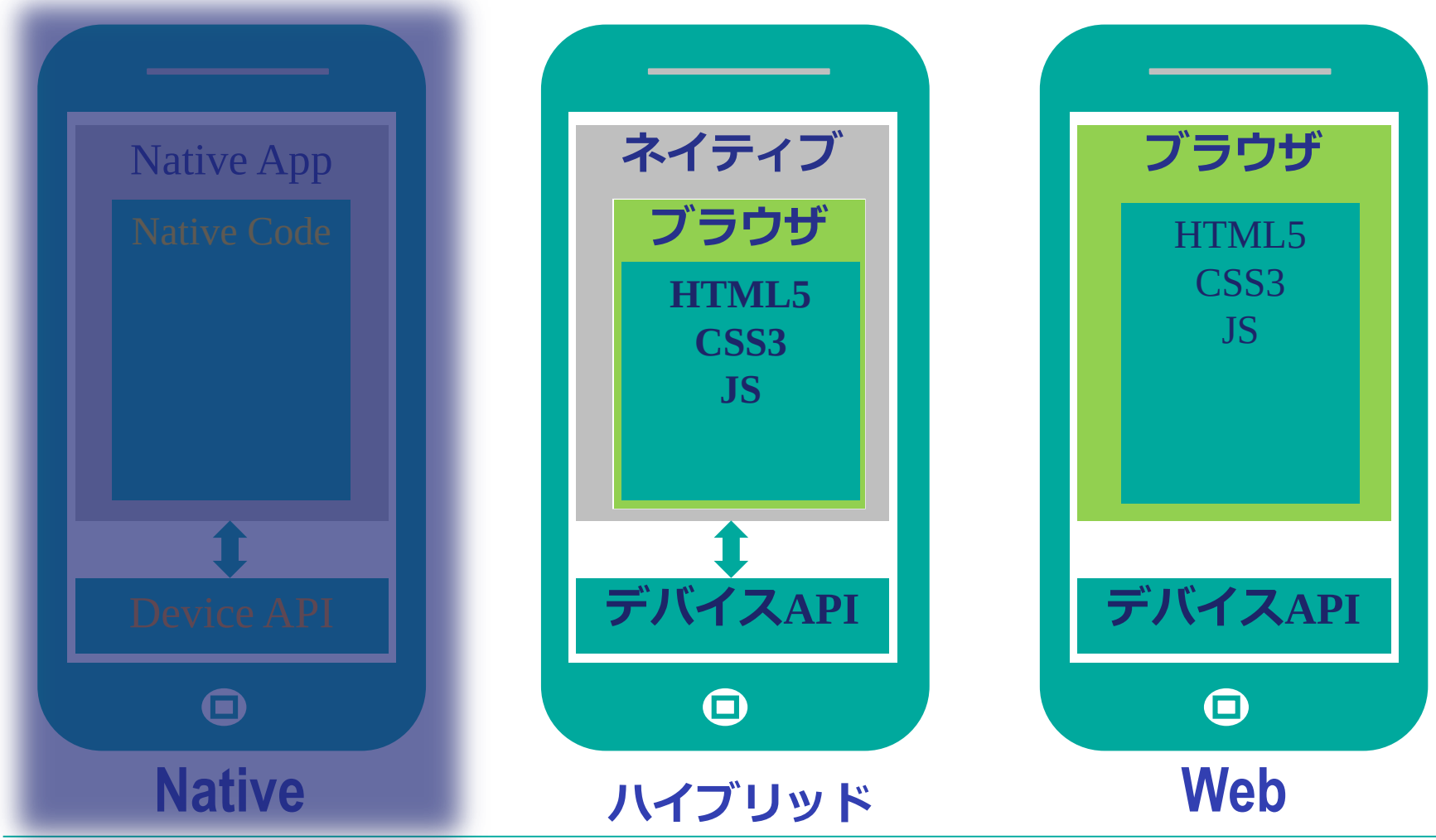


 |

モバイルアプリケーションを開発する方法



Zen Mojo: Webとハイブリッド



 |

モバイル用Zen Mojo

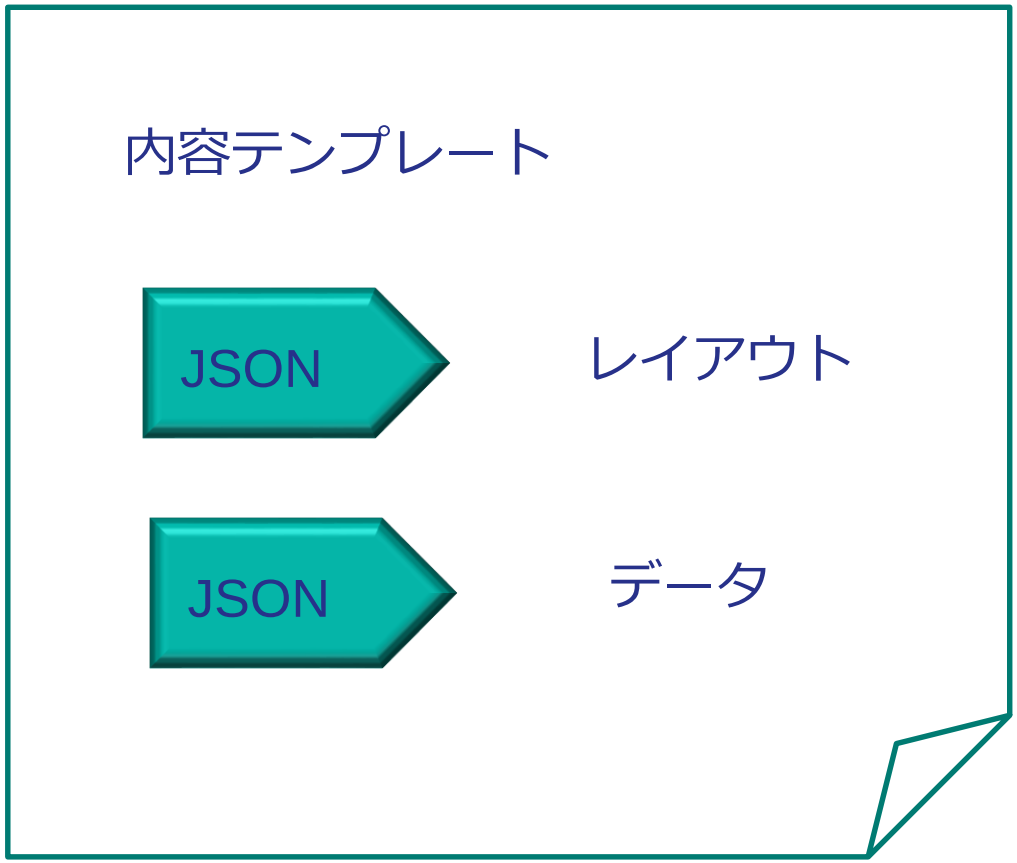
Zen Mojoアプリの開発方法

Zen Mojoでの開発は、Zenとは根本的に違っている

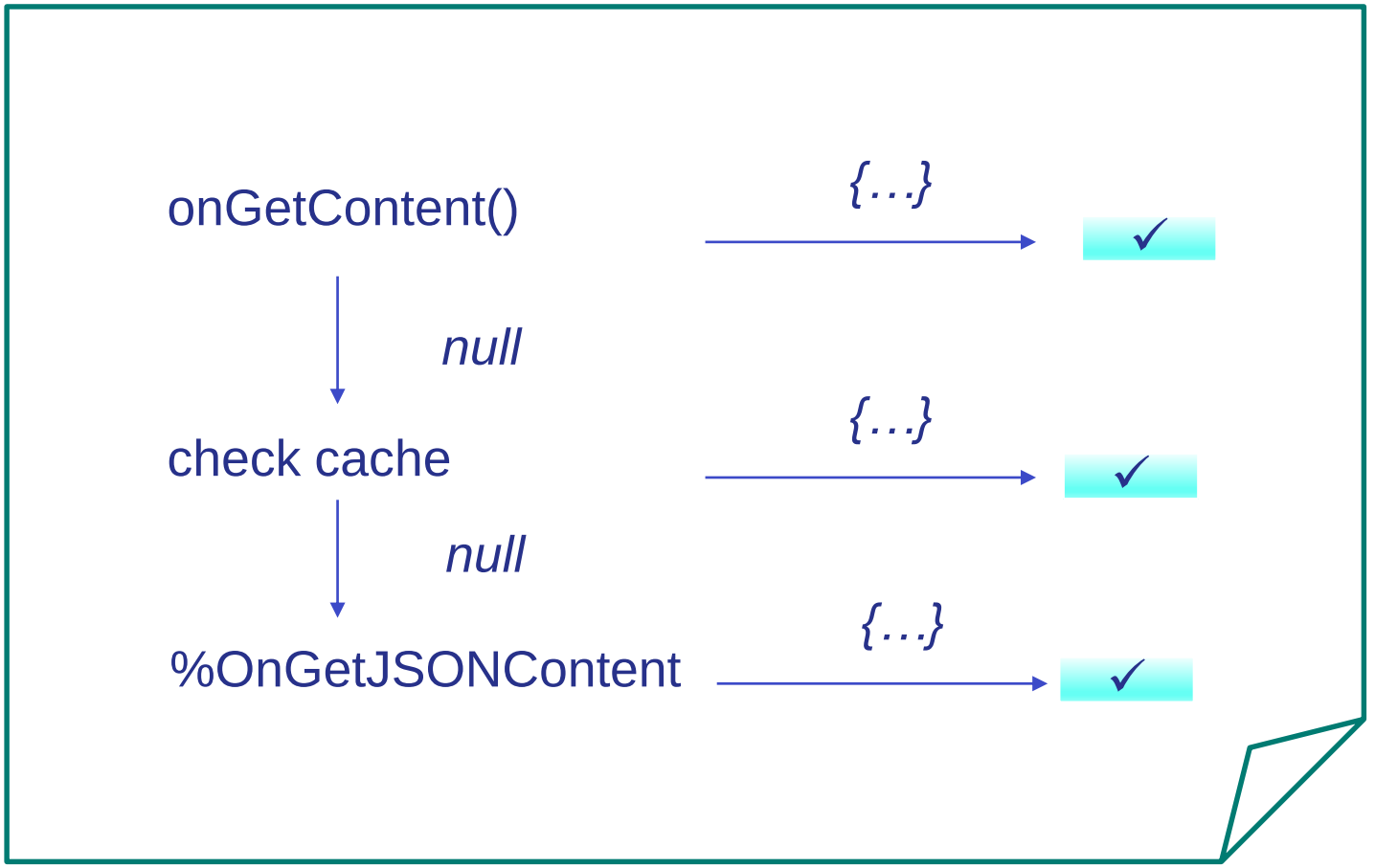
- シングルページアーキテクチャ
- クライアント側での描画
- より進化したキャッシング機構
- 効率の良いクライアント／サーバー通信(JSON)
- 完全データ駆動 (レイアウトとデータの内容)

Zen Mojoアプリの開発方法

Zen Mojoアプリの開発方法



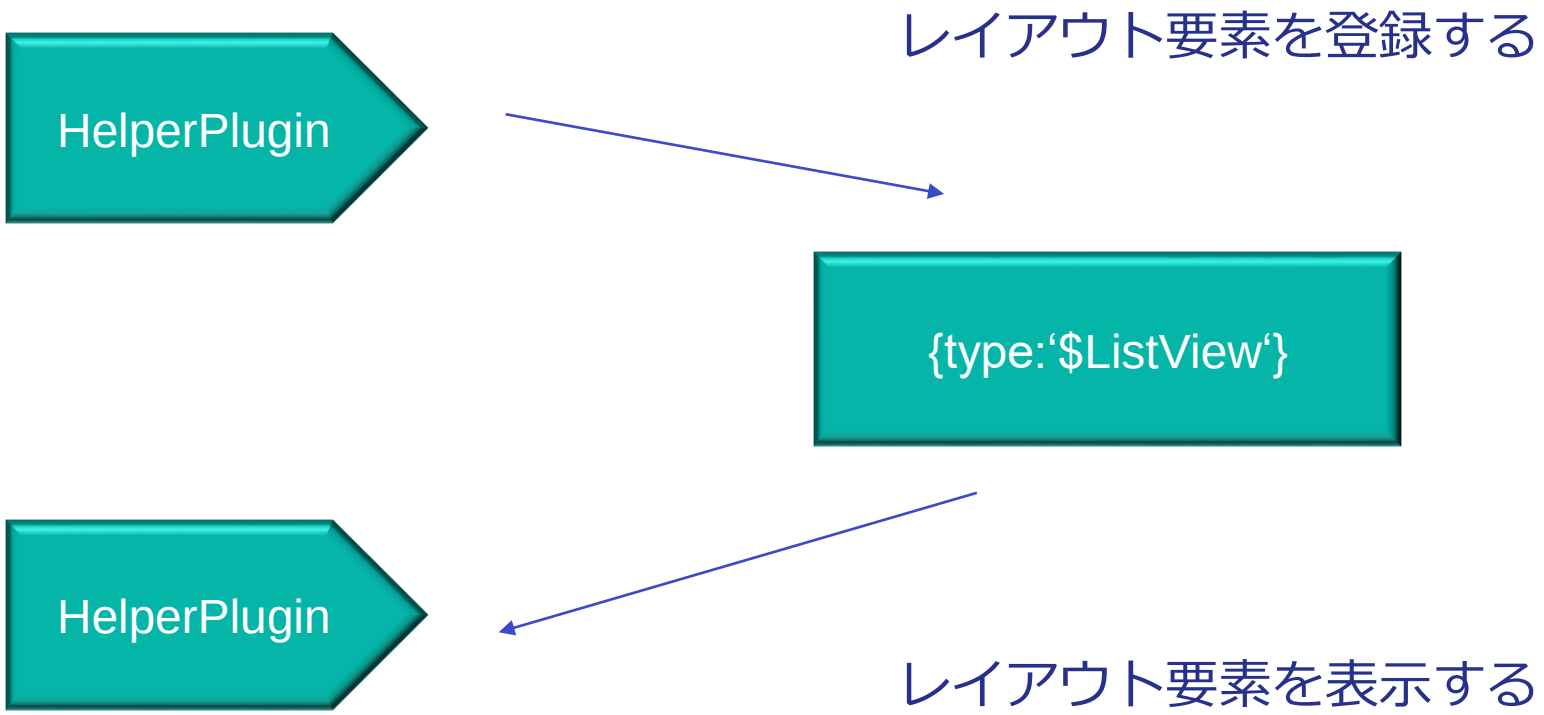
Zen Mojoアプリの開発方法



 |

Zen Mojoアプリの開発方法

Zen Mojoアプリの開発方法



Zen Mojoアプリの開発方法

<documentView>
ページマネージャー
ヘルパープラグイン
</documentView>

全てのイベントは、documentView
コンポーネントによって起こされ、
その内容テンプレートに委譲される

イベントハンドラー

- onchange
- onselect
- onevent

 |

コードの例

 |

Zen Mojoアプリの開発方法

HomePageの定義

```
/// Created using the page template: Default
Class coffeeDesktop.HomePage Extends %ZEN.Mojo.basePage


/// List of JSON providers created for this page.
/// Used by contentProvider.
Parameter PROVIDERLIST = "data,layout";
```

```
/// Starting template class for this page
/// Used by contentProvider.
/// This is overriden by application settings, of provided.
Parameter TEMPLATECLASS = "coffeeDesktop.coffeeDesktopTemplate";
```

```
/// Comma-separated list of additional JS include files for the page.
Parameter JSINCLUDES As STRING = "dojo-release-1-9-1/dojo/dojo.js";

/// Comma-separated list of additional CSS include files for the page.
/// If this page is using CSS3 then the CSS3INCLUDES parameter is used (if defined).
Parameter CSSINCLUDES As STRING = "dojo-release-1-9-1/dijit/themes/claro/claro.css,
                                  dojo-release-1-9-1/gridx/resources/claro/Gridx.css";
```


Zen Mojoアプリの開発方法

単純なXData 内容ブロック

```
<mojo:documentView id="main
  initialDocumentKey="custome
  initialLayoutKey="customer-list"
```

templateクラスのonGetContent()にディスパッチする

```
  ongetlayout = "return zenPage.getContent('layout',key,criteria);"
  ongetdata = "return zenPage.getContent('data',key,criteria);"
```

```
<mojo:dojo-1.9.1-PageManager onPageShow="zenPage.onPageShow(key);">
   <mojo:dojo-1.9.1-DijitHelper/>
   <mojo:dojoGridX-1.3.0-Helper/>
   <mojo:HTML5Helper/>
   <mojo:mojoDefaultHelper/>
 </mojo:dojo-1.9.1-PageManager>
</mojo:documentView>
```

```
</mojo:documentView>
```


Zen Mojoアプリの開発方法

template内にコールバックを実装中

```
/// Client-side content provider method.
/// <var>which</var> is the name of the content to return.
/// <var>key</var> is the key value associated with the request.
/// <var>cr
ClientMetho
    which,
    key,
    criteri
{
    switch (w
        case
            r
        case
            return null;
    }

    return null;
}
```

```
switch (which) {
    case 'layout':
        return this.getLayout(key,criteria);
```

```
case 'data':
    return null;
```

サーバーの%OnGetJSONContent()にディスパッチする

 |

Zen Mojoアプリの開発方法

クライアントにレイアウトを生成中

子オブジェクトの配列

```
content = {
    children:[
        {type:'$LayoutContainer',
            key:'layoutContainer-1',
            design:'headline'
    {type:'$ContentPane',region:'top',title:'Headline',key
        {type:'$div',$content:'Customer List',cssClass:'he
        {type:'$MenuBar',children:[
            {type:'$MenuBarItem',label:'Customer List',
              key:'menu-show',
              value:'customer-list'},
```

hildren:[

customer-list'},

Typeプロパティにクライアントコンポーネントのタイプを記述する

Zen Mojoアプリの開発方法

サーバー側に内容を生成中

```
set pObject = ##class(%ZEN.proxyObject).%New()
i
```

```
"customerListColumns":[ {
                        "field":"id",
                        "name":"Identity"
            }, {
                        "field":"name",
                        "name":"Customer"
            }
]
```

```
        set column = ##class(%ZEN.proxyObject).%New()
        set column.field = "name"
        set column.name = "Customer"
        do pObject.customerListColumns.Insert(column)

    }
```

 |

RESTとは

- **ロイフィールディングが提唱したウェブアプリケーションのアーキテクチャ上のスタイル**
  - “表現上の状態の転送をよいウェブアプリケーションの振舞のイメージを喚起することと想定する: リンクを選択すること（状態遷移）でアプリケーションが進行中のウェブページ（仮想状態マシン）の次のページに移動し（アプリケーションの次の状態を表現しながら）、ユーザーに橋渡しされ、ユーザーの利用に合わせて表現される ”

さらにいいことには

"RESTは、コンポーネントのやりとりのスケーラビリティ、インターフェースの一般性、コンポーネントの導入非依存性、中間コンポーネント間のやりとりの遅延の削減、セキュリティの強制、レガシーシステムのカプセル化を強調している"

- Webopedia

 |

REST

RESTは標準でもプロトコルでもなくて、アーキテクチャー上のスタイル

RESTは、既存のウェブ標準であるHTTP、URL、XML、JSONなどを使う

RESTはリソース指向
リソースまたは情報の断片をURIで指定し、サーバー/クライアント間の双方向に渡される

RESTの原則

一定のインタフェース: 簡潔にアーキテクチャーに紐づけない、その結果各部分は独立に進化する

ステートレス: クライアントのコンテキストは要求間でサーバーに保存しない
リクエストをサービスするために必要な情報はすべて毎回送る

キャッシュ可能: よく管理された部分的および完全なキャッシングがいくつかのクライアント/サーバー間のやりとりを削る
スケーラビリティと性能を改善する

RESTfulウェブサービス

RESTfulウェブサービスというのは、HTTPとRESTの原則を使って実装したウェブAPI

URIのようなディレクトリ構造で識別するリソースの集合

(https://www.googleapis.com/calendar/v3/calendars/*GlobalSummit*/events)

操作は、明示的にHTTPメソッドを基礎とする(GET, POST, PUT, DELETE)

情報は、インターネットのメディアタイプ、通常はJSONに基づき転送
他のタイプにはXML,HTML, CSV (テキスト)が含まれる

CRUD操作

REST操作はhhtpプロトコルメソッドで定義されている４つにタイプに集約される:

REST	HTTP	
Create	Post	POST https://api.twitter.com/1.1/statuses/retweet/241259202004267009.json
Read	Get	GET https://api.twitter.com/1.1/statuses/user_timeline.json?screen_name=twitterapi&count=2
Update	Put	PUT https://www.googleapis.com/calendar/v3/calendars/calendarId/events/eventId
Delete	Delete	DELETE https://www.googleapis.com/calendar/v3/calendars/*calendarId*/events/*eventId*

REST優位性

REST

- 簡潔性 (使用、保守、テストが簡単)
- 表現のたくさんな選択肢がある(JSON, CSV, HTML, XML)
- 人間が可読できる結果
- 性能
  - スケーラブルアーキテクチャ
  - 軽量要求と軽量応答
  - より簡単な応答の解析
  - 帯域の削減 (キャッシング、条件付GETなど)
  - JSON表現を使うとクライアントに適している

REST優位性

Soap要求

```
<?xmp version=“1.0”?>
<soap:Envelope xmlns:soap=
   soap:encodingStyle=                                    >
<soap:Body ord=                                  ”>
   <ord:GetOrderDetails>
      <ord:OrderNumber>12345</ord:OrderNumber>
   </ord:GetOrderDetails>
</soap:Body>
</soap:Envelope>
```

REST要求

http://www.igroup.com/order?ordernum=12345

 |

URL / URI

RESTインターフェースは URL/URI で定義する

URI – Uniform Resource Identifier

- ネットワーク上の特定のリソースを識別
- 例: http://www.igroup.com/order

URL – Uniform Resource Locator

- ネットワーク上のリソースの特定の表現方法のアクセスを提供
- http://www.igroup.com/order?ordernum=12345 または
- http://www.igroup.com/order/ordernum/12345

セキュリティ

セキュリティはインターフェース開発者にゆだねられる

- RESTには予め定義済メソッドはない

ウェブアプリケーションとして既に利用可能なものをおおいに利用すべし

- SSL/TLS（https:）
- OpenId Authorization (Oauth)
- Hash-based Message Authentication Code (HMAC)

Cachéでの実装

2014.1に新クラス - %CSP.Rest

SMP上でディスパッチクラスを登録
RESTアプリケーションベースURLとマッチングする

- システム>セキュリティ管理>ウェブアプケーション>ウェブアプリケーションの編集
- 新規ウェブアプリケーション
- /csp/samples/globalsummit
- Dispatch Class: Rest.Broker

UrlMap Xdataブロックを使ってリクエストをＨＴＴＰ操作とターゲットのクラスメソッドに引き渡す

- XData UrlMap {
  <Routes>
  <Route Url="/employee/html/list" Method="GET" Call="Rest.HTML:GetAllEmployees"/>
  </Routes>}

REST vs. SOAP

REST	SOAP
１つのスタイル	標準
“適切な” RESTとしては、トランスポートにはHTTP/HTTPSが必須	普通はトランスポートはHTTP/HTTPSだがほかのものでもよい
応答データは通常XMLやJSON形式で転送される 平均的にはJSONのほうが軽い（SOAPヘッダーのオーバーヘッドがない）	応答データはXML形式で転送される

REST vs. SOAP (続き)

REST	SOAP
要求はURI形式で転送 •SOAPに比較してかなり軽い •長さに制限あり •入力フォームフィールドを簡単に使用可能	要求はXML形式で転送
メソッドとURIを解析するとその意図がわかる	意図を理解するにはメッセージペイロードを解析しなければならない
	WS* イニシアティブが圧縮やセキュリティのような課題の改善に取り組む

REST vs. SOAP (続き)

REST	SOAP
JavaScriptから呼出し簡単	JavaScriptはSOAPを呼び出すことは可能だが、難しく洗練されたやりかたではない
JSONが返ってくると、非常に強力	JavaScriptのXML解析は遅くて方法がブラウザ毎に異なる

REST/JSONは以下のようなケースに最適…

限られた帯域とリソース

- 開発者定義の構造の柔軟性
- どのブラウザも利用可能

完全にステートレスな操作

- 例えば、ステートレスなCRUD操作

キャッシング状況

- RESTアプローチは情報がキャッシュできるときに非常にうまく動作する

SOAP/XMLは以下のようなケースに最適 …

非同期処理、非同期起動

- SOAPは保障できるレベルの信頼性とセキュリティを提供

正式な契約

- SOAPはプロバイダーとコンシューマ間の交換の厳密な仕様を与える

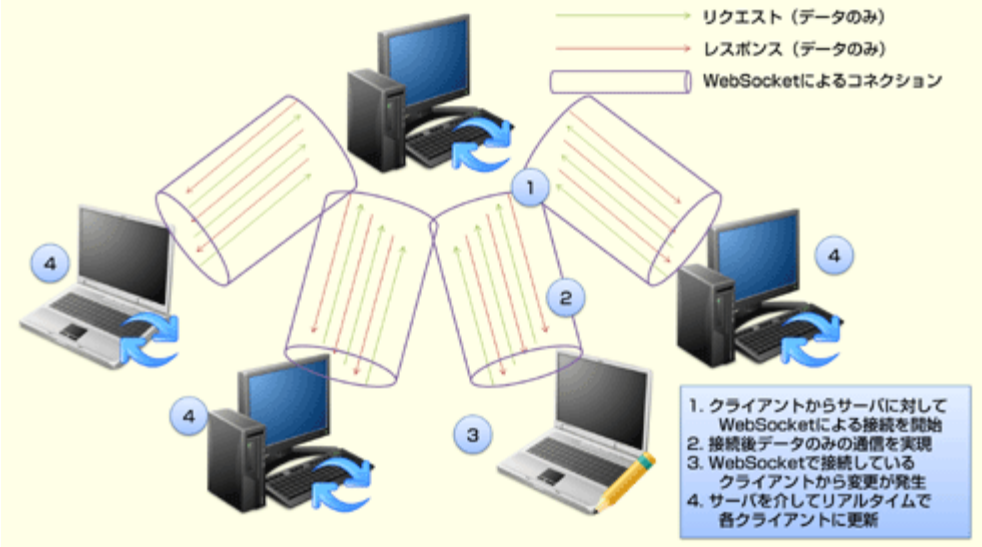
ステートフル操作

- SOAPは、コンテキストと会話状態管理をサポートする追加の仕様を持っている

WebSocket

サーバとクライアント間は一度でも接続が確立すると、明示的に切断しない限り通信手順を意識することなくデータのやり取りをソケット通信で実施できる

WebSocketで接続が確立しているサーバとすべてのクライアントは同じデータを共有し、リアルタイムで送受信できる



ウェブ関連最新技術動向

インターシステムズジャパン株式会社

 |