Php 刘海涛

Php：

简介

Php7

Php web请求的生命周期

Php语法：

<?php...?>

关键字大小写不敏感，变量大小写敏感

Echo与print的区别

全局变量使用：直接使用变量名/使用globals数组（函数内部需要指明用到的global变量）

类型自动转换：隐式/显式（(type) variable）

空串NULL

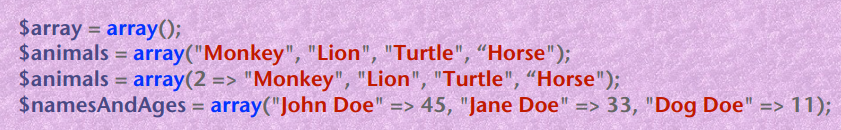
字符串的转换

单引号和双引号的区别（双引号可以解释里面的变量用值代替）

Array：

Indexed array

关联数组



Asort（） 按值进行排序

Ksort（） 按键进行排序

函数：

分类

1. 内置：array\_push等
2. 用户定义

Function functionName() {

Doing:

}

弱类型：参数没有类型，返回值没有类型，没有显式的return时，返回一个null

函数不支持重载

严格类型：

默认情况下php会把错误类型的值强制转换为期待的值

Php7：TypeError

Declare(strict\_types=…)

返回类型声明：

Function sum($a, $b):float{

…

}

返回类型为float

函数调用：若实参个数多于形参，忽略后面的

若实参个数少于形参，也能执行但是会有warning？？？？？

可以使用默认参数值

Function functionName( parameter = 50) 非默认在先，默认在后

可变长度的参数列表

Function sum(…$numbers) （php5.6及之后版本）

Function sum()

匿名函数（闭包）：

????????

<http://www.php.net/manual/zh/functions.anonymous.php>

可以再php中输出html的语句，但是不建议，因为字符转义很麻烦，且php语言中要尽量减少嵌入的输出语句

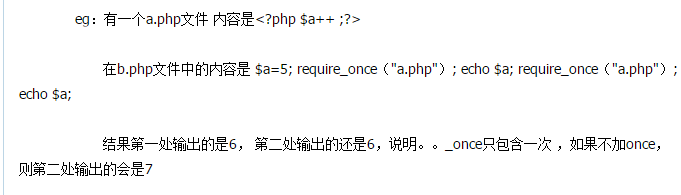
如何在网页中动态插入？

引入外部文件：鼓励模块化思想，提升代码可重用性

Include(“header.php”)

Include和require的差别：当找不到目标文件时，include给出警告，require跑出错误

Include\_once加once后，系统会进行判断，如果已经包含，则不会再包含第二次



Lecture8 more php for serverside programming

提交数据

Get&post区别：

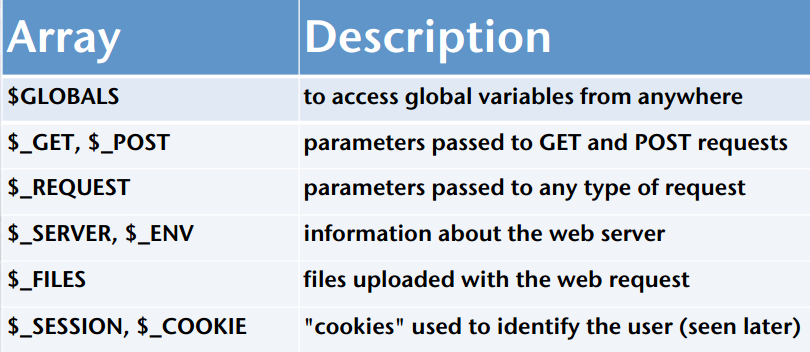
Chapter8 page4（偏爱post）

url编码：

表单处理：

1. 超全局变量：无视变量作用域，php中预定义的一些变量，可以从任何地方获取

$\_GET, $\_POST传递到get和post请求的参数



Request 按照 get post cookie 的顺序传递 ，可能导致匿名篡改问题？？？？

处理表单数据：

Inport\_request\_variables(“gp”, “rvar\_”) 第二个参数是一个前缀

Htmlspecialchars() 把特殊字符转换为html实体如<>转换为&lt; 和 &gt; 可以防止攻击

表单验证：确保表单的值都是正确的

避免空白值

保证数据类型（电话号、邮编、信用卡）

确保正确的格式及值的范围

确保值之间没有冲突

正则表达式：能用string用string，不能的用正则（高功能文本匹配）

/.匹配任意一个字符

/i大小写不敏感

()分组

^匹配开头 $匹配结尾

\作为转义字符

Eg: <\/b> 正则表达式：/[\\\/b](file:///\\\/b)/

{min,max} 最小出现次数~最大出现次数

{3}恰好出现3次

{,6}最多出现6次

[]字符集，n选1

面向对象的PHP

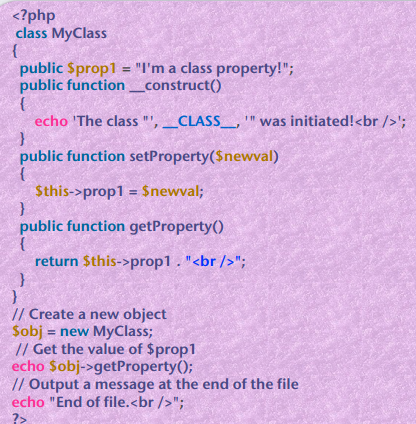
构造函数：低版本，高版本不建议

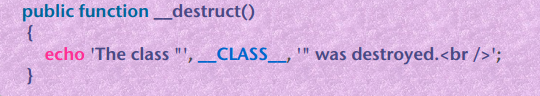
魔术方法：高版本，php内部提供，提供与名字相关最简单的实现但是有特定意义，在使用对象的时候更加容易

Eg:\_\_construct(), destruct(), \_\_call(), \_\_callStatic(), \_\_get(), \_\_set(), \_\_isset(), \_\_unset(), \_\_sleep(), \_\_wakeup(), \_\_toString(), \_\_invoke()，\_\_autoload()

自定义函数不要用\_\_开头

魔术常量：\_\_ CLASS\_\_,PHP内部定义





\_\_toString()转换成string

Php结构化比较厉害

使用面向对象的原因：1.易于实现，可复用

1. 更好的组织，打包比较方便

抽象类与接口：面向接口编程，组合优于继承

Chapter9 php framework

File inclusion –> template engine -> MVC framework

模板引擎

MVC

Model，View，Controller

Web中的应用：Ruby on Rails（RoR）

Model：数据存取（抽象的）

View：单个模型支持多个视图（HTML模板）

Controller：处理和响应事件

MVC优点：

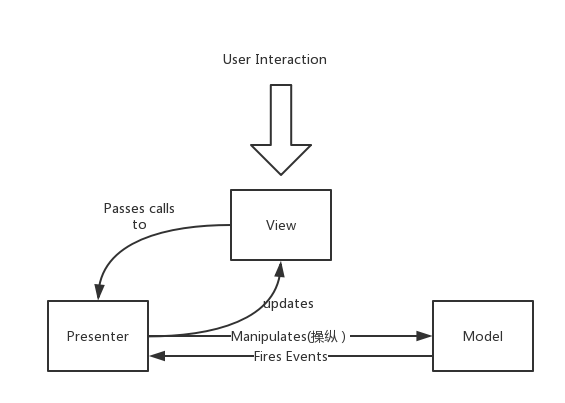
更好的体系结构设计

易维护

易扩展

变种：MVP(P: presenter)：视图与模型之间没有直接通信

mvvM

MVVM Presenter改为ViewModel

updates改为fires events

web框架

软件框架

支持动态网站开发

用标准方法解决标准问题

框架和库的差别：框架是一个整体，库可以直接调用其中的某一个方法

使用框架的好处：

模板化

对功能的抽象

利于快速开发

Index.php URL解析

。。。

Zend framework chapter 9 page 25

Zf：面向对象；松耦合（按需使用，组件之间依赖性较弱）

Lecture11 web performance

网站速度越快，越易于爬取、访问、更易于排序

WPO：web performance optimizatioin网站性能优化

实现：影响因素：延迟和

延迟：传播延迟、传输延迟（把 分解到当前链路上）、处理延迟（）、排队延迟

提升方法：①考虑延迟和带宽对web性能的影响

②TCP的传输协议对HTTP的影响

③HTTP协议本身的特点和缺点，无法避免

④web应用的发展趋势和性能需求

⑤浏览器的局限和优化



HTML-> Dom –>Render

CSS -> CSSOM->Render

Render: –>Render Tree -> layout ->paint

JS会动态改变DOM结构，改变CSSOM

PLT网页加载时间

Web程序包括三个任务：获取源文件，网站布局和渲染，js执行

更多的带宽不一定有用

瓶颈是延迟

定量感知web性能 lecture11 page9

消除或减少不必要的网络延迟

减小传输字节的数量

性能原则：

1. 尽量减少DNS查找
2. 重用TCP连接
3. 最小化重定向（重定向：至少经过一次重新申请的过程，不利于对网页价值的提升）
4. 内容 网络(Content Delivery Network CDN)

离用户越近，每个TCP请求的网络延迟就越少‘

1. 削减不必要的资源
2. 在客户端缓存数据
3. 压缩传输内容
4. 消除不必要的请求字节

部署双协议，不同的协议交付不同的版本

动态HTTP1.x 和HTTP2.0优化

#php 刘海涛

@(chapter11)[php\_performence]

YSlow规则：

- 最小化HTTP请求

- 使用一个CDN服务

- 避免空的src或者href

- 添加Add an Expires or a Cache-Control Header

- 压缩css和html文件

- 把css放在头部

- 把scripts放在最后

- 避免css表达式（每次的轻微操作都会使得这个css表达式重新执行）

- 外部引入js和css（利于维护）

- 减少DNS查询（DNS查询：）

- 最小化js和css

YSlow rules2:

* 避免重定向
* 移除重复脚本
* 实体标签
* make ajax cacheable
* 使用ajax请求的时候用get请求
* 减少dom标签数量
* 减少cookie大小
* 使用无cookie的域来请求资源
* 避免过滤器
* 避免图像在html中缩放
* 避免filters
* 在根目录下放置小于1k的favicon.ico文件并进行缓存

**script加载方式：defer vs async**

* 延迟：加载js文件和扩展dom树是并行的，在加载完之后才执行
* 异步：加载和执行相对于dom树扩展是乱序的

str+=’wb’+’cd’（先创建临时变量保存wb+cd的结果再与str连接）   
str=str+’wb’+’cd’（不需要创建临时变量）

**使用类中的常量**

编译时解析，没有执行负载   
更小的哈希表，更快的查询   
命名空间&更短的哈希名称   
简洁的代码会加速调试

**require vs. require\_once**

require性能更高

### XML、JSON

#### XML

简介：

* xml: extensible markup language
* 与html的不同：设计用于存储和传输数据
* html：结构+展示内容
* 自描述的（self-descriptive)

xml vs. html：   
xml：数据存储传输、无预先定义的标签、面向计算机   
html：面向内容、标准化标签、面向人

xml用于数据交换：

* 基于文本格式
* 对软硬件都没有依赖
* 在不兼容系统之间传递数据
* 可以被多种应用读取

一个xml文档必须有一个根节点   
将日期放在属性中，无法分解为树节点=>尽量使用标签替代属性

XML的优缺点：   
pro：

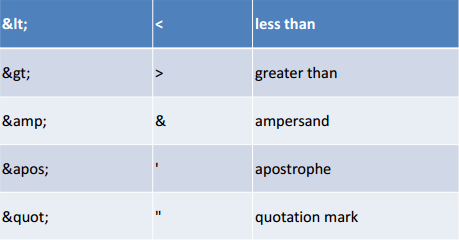
* 易读性
* 标准化格式使得自动化更加容易
* 标准的、独立的、开元的平台
* 集合可以表示所有普通类型的数据

con：

* 冗余，降低性能
* 很难将数据放到一个好的xml格式中

建议：数据使用标签，元数据用属性

转义字符：

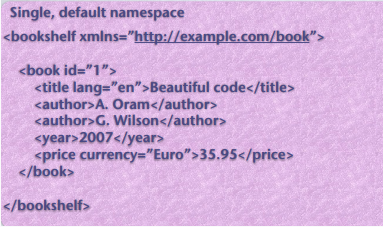


### CDATA

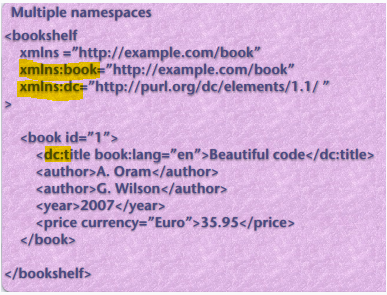
character data   
语法：<![CDATA[ …]]>   
不被解析器解析   
用途：避免被转义   
注意：不能嵌套   
解决嵌套问题：base64 encode()   
base64 decode()

命名空间：   
避免不同xml源文件中名称的冲突

* s使用单一的、缺省的命名空间



* 使用多个命名空间



#### XML in php

##### XML Parser

###### 基于树的解析

把完整文档转换成树状结构（借助DOM形式，需要先全部加载文档到内存再转换）   
应用：对于比较小的文档比较使用

###### 基于事件的解析

把当前文档看做一系列事件，基于事件驱动的（不需要完整加载到内存，读取到哪个节点处理哪个事件）   
应用：适用于比较大的文档   
例子：XML Expat Parserr   
XML Expat Pars

##### XMLReader/-Writer

基于push/pull   
不把xml全部加载到内存中

#### XPath

##### 简介

对当前xml文档的节点和属性进行提取   
应用领域：

* XML样式语言转换
* 编程语言中对xml节点进行提取   
  + /bookshelf/book
  + //book
  + 以上两种方式都能拿到所有的book列表

##### 性质

* 寻址   
  + 绝对路径   
    /bookshelf/book/title   
    /bookshelf/book/author
  + 相对路径   
    book/author   
    ../title
* 上下文   
  基于节点的相邻关系来进行节点的选择
* 属性   
  选取属性的方式@   
  /bookshelf/book/price/@currency
* 步骤   
  借助于上下文环境，对于父代反向导航和，子代进行导航
* 基于索引的方式   
  []取值从1开始   
  /bookshelf/book[2]
* 通配符   
  \*   
  /bookshelf/\*/title

##### xPath语法：

轴(axis)：跟当前路径节点的关系   
节点测试(node tests)   
predicate谓词

<axis>::<nodetest>[<predicate >]

轴语法：

* 普通语法
* 祖先语法
* 同代
* 下一个同代（following-sibling)
* 命名空间

谓词语法：

<nodetest >[<predicate >]

Eg:Select all books with Id 1: book[$id=’1’]

### JSON

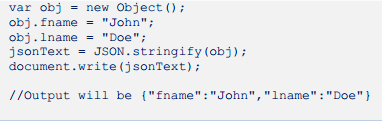
javascript object notation   
语法

* json使用name/value对来存储数据
* ，用于分隔不同的数据值
* 使用{}来界定对象
* []用于存储数组

JSON vs. xml

* 相同点   
  + 可读性，自描述性
  + 都代表了分层的结构，数据中可以包含数据
  + 都可以被各种编程语言获取和解析
  + 都可以被一个XMLHttpRequest object获取
* 不同点   
  + XML需要XMLparser， json使用js方法解析
  + json中不需要标签
  + json更短，更易读
  + json可以使用数组

如何使用json读取数据   
js方法，JSON.parse(jsonData)   
将数据转换为json数据   
JSON.stringify(obj)



# Ajax

## RIAs(Rich Internet Applications)

Web Application基于CS(client-server)胖客户端

BS架构：瘦型客户端

缺点：interface= html+css+js+images

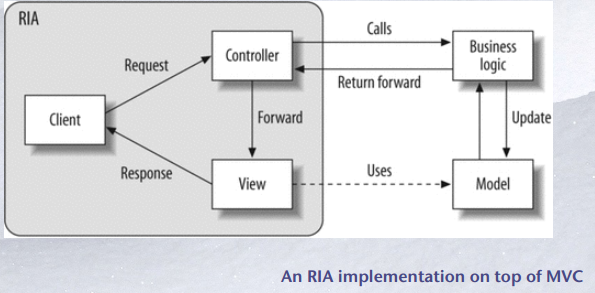
表现形式不够丰富（没有video（需要插件），对矢量图没有支持）

GUI动态交互不够自然

响应较少

RIA：胖型客户端 富互联网应用、富数据模型

在BS架构中体现桌面应用的用户体验



计算模式前移：由前端来分担一些计算 AngulaJS

RIAs优点：不需要安装、易于升级，富UI，更多的响应式UI、Client-Serrver平衡，异步通讯，网络效率（允许获得片断数据）

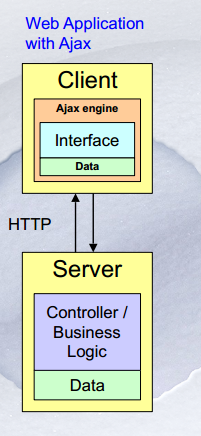
缺点：SEO搜索引擎（对SEO不友好）、整体性缺失、RIA架构和Web

RIA实现：

插件式：Flash, JavaSwing, Silverlignt等，交互性更好，但是入门门槛高

非插件式：Ajax

Ajax传统实现方式：

客户端通过js异步发送消息

优点：

①不需要安装，更新更容易

②不易受到攻击

③易于获取

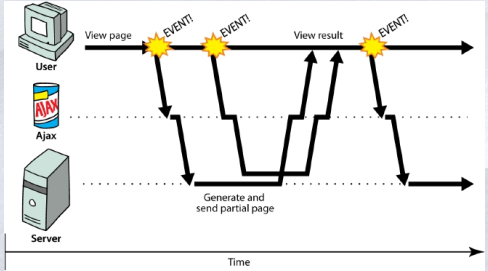
最大的优势：基于web，是标准的

最大的劣势：js被用户关闭，

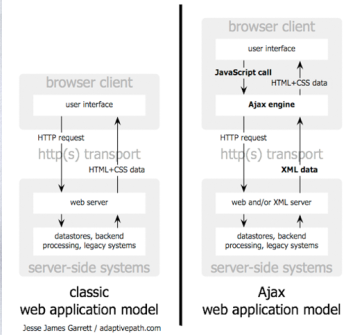
同步通讯：每个请求都要在一些操作完成后才能发送，客户端在服务器完成响应之前没有响应（瘦客户端的劣势）

## Ajax

异步性



Ajax引擎作为媒介，在用户和server之间进行数据交换（交换数据的格式：文本、xml、json）



OutLook：（伪）实时更新信息—第一个使用Ajax的

Gmail：容量大

Ajax：Asychronous JavaScript and XML

是一个技术集

在后台从服务器上下载数据

把用户的同步行为变成了用户与UI的交互（动态 局部更改）

避免了“点击——等待——刷新”模式

优点：更好的交互性和响应性

浏览器交互性能更好

局部刷新的特性会减少和web服务器的联系

节省带宽（局部加载）

减少网络交通

缺点：前进后退浏览历史出现问题

书签（如何分辨局部刷新前后的差别）难以使用

需要js参与工作

网络延迟，会降低可用性

如何改善和SEO之间的关系

应用：搜索框自动完成

级联下拉框

实时交流（如即时通信）

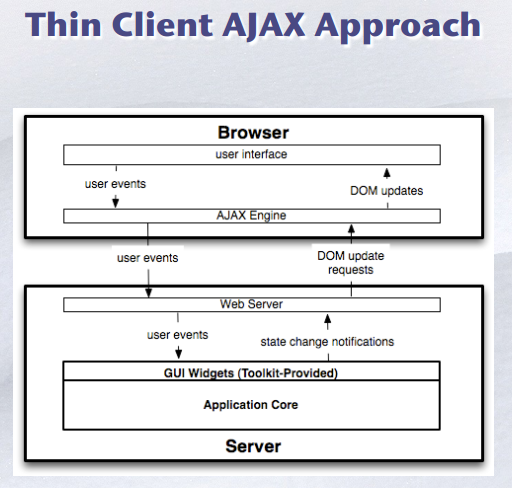
实时数据更新（如分数更新）

审批反馈

自动保存用户信息

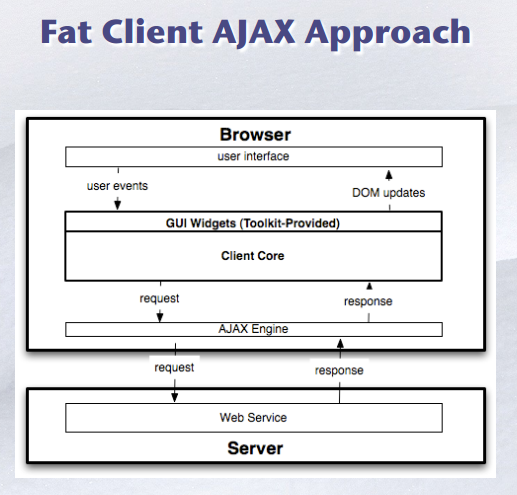
类似于web socket

Ajax瘦型客户端：



优点：更简单的莞城工作（忽略网络）

胖型；



瘦型Ajax：chapter13 page47

## XMLHttpRequest

Components of Ajax

Js：弱类型语言，提供Ajax的支持

Dom：对文档的操作

Css

XMLHttpRequest：js对象提供数据在后端与服务器之间的通信，利用回调的方式对网页进行更改（Dom）

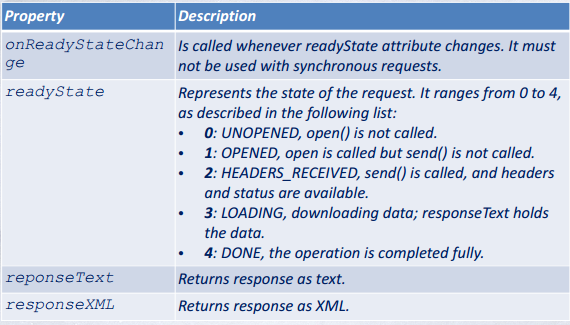
Xml/Json：数据格式

Ajax处理过程（事件驱动）：

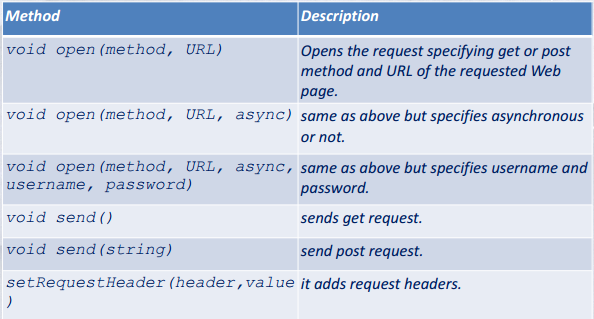
先构建一个xmlhttprequest对象，对象呗配置好之后，向服务器发起一个异步请求，

XMLHttpRequest Objext函数

onReadyStateChange：当readyState属性改变时被调用，



方法



新建一个XMLHttopRequest方法：

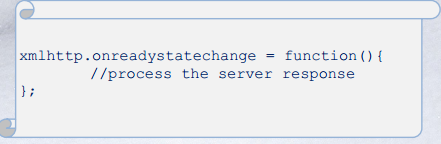
Variable = new XMLHttpRequest();

对于IE6：variable = new ActivXObject(“Mocrosoft.XMLHTTP”)

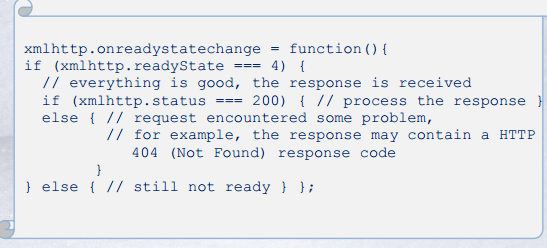
考虑浏览器兼容性



对请求响应的处理：

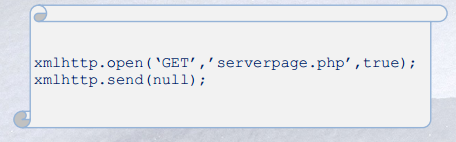


Eg：



Xmlhttp.status =304缓存

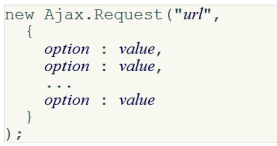
配置发送

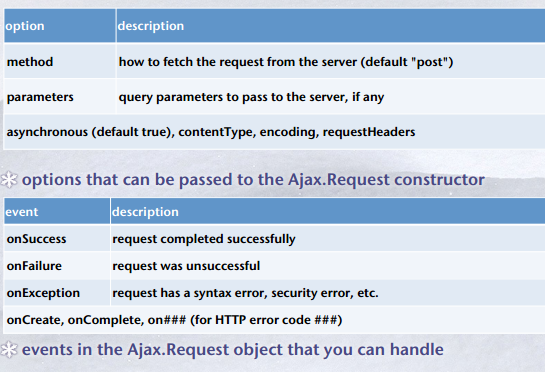


## Ajax in Prototype

原型Ajax模型

构建一个ajax请求

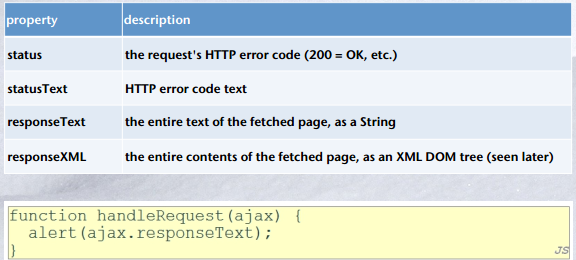




Eg：



Ajax response objct

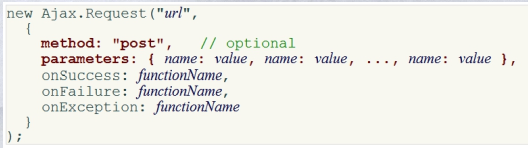


Ajax updater

获取一个文件并把它的内容插入到innerHtml中



Post



## Limits of Ajax

## Debugging Ajax

代码优化：

优化js代码

性能瓶颈：对dom的访问，而不是对js的优化

Ajax实现的实践：

充分理解

合适场景中使用ajax

学会使用原生的ajax

Ajax库的限制：

性能、功能的牺牲（readystate的状态）

数据格式：

Xml：（不能实现高性能的ajax）

互操作性好

严格规范化

易于确认

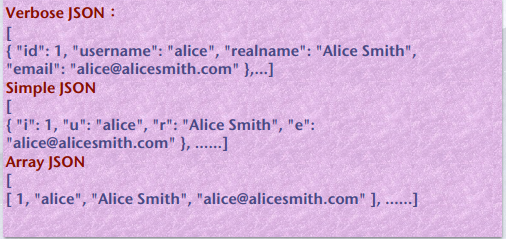
冗余

歧义

解析模糊

（XPath：兼容性不够好）

JSON



JSON-P（JSON填充）

Json文件格式与js的相似性演化而出

JSON-P与JSON不同，JSON是一种简化的轻量级数据格式，JSON-P是一种非规范的数据交换方法

特点：支持跨域

插入动态脚本的标签，将json数据当作js的原生对象进行处理

客户机和服务器之间的解耦

HTML

Custom Formatting（自定义格式）

按照一定的分隔符，将想要传递的数据传递过来的形式

数据格式总结：

偏爱轻量级的方式（JSON-P或自定义格式）

大数据量：自定义格式更快

小数据量：JSON\_P更快

缓存数据（提升性能）：

浏览器端和服务器端缓存的使用

### Lecture14 Web服务

#### Web服务

定义：用于支持网络上软件可协同交互的

通过给定的web接口可以调用API，利用HTTP的方法请求互联网中http的功能

常见web服务

RPC-远程服务调用

SOA-面向服务的体系结构

REST-表述性状态转移

RPC-remote procedure calls

允许当前进程调用位于异地的进程

客户机-服务器的模式：客户机向远程进程发出调用请求；服务器获取参数并在本地执行进程，之后把结果传送给客户机；客户机收到结果

根据数据传输格式分为两种RPC方式：XML RPC和JSON RPC

SOA面向服务的体系结构

一个应用，分成不同的功能单元，功能单元之间定义良好的契约

使用注册中心：服务提供者的服务对外open，服务请求中心获取服务的地方

SOAP：simple object access protocol简单对象访问协议

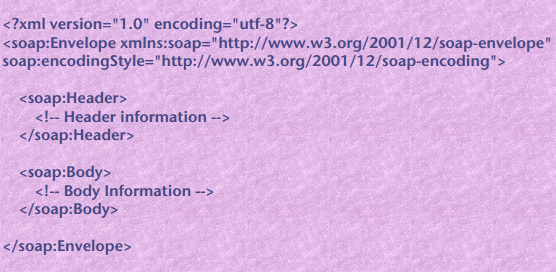
WSDL : web services definition/ description language（web服务定义）

UDDI：universal description, discovery, and integration

SOAP:简单对象访问协议

基于XML的简单的轻量级访问协议，一种用于描述业务逻辑所使用的交换信息的形式

SOAP基本骨架



WSDL：web服务描述语言

基于xml构建的协议，用于描述web服务（可用的公用方法，以及服务地址）

能够找到对应服务对应方法调用的状况

SOA发现资源需要额外及值（注册中心）

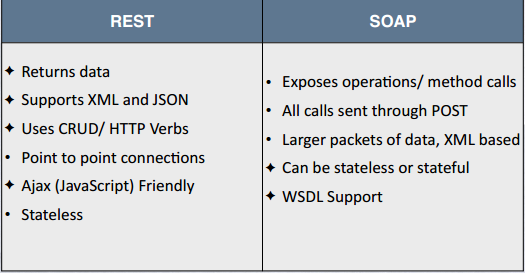
REST：表述性状态转换

通常网站会提供rest和soap两种API

依赖于轻量级协议的无状态通信

不需要额外的机制来发现资源（与SOA相比）

设计原则：把网上的所有事物都抽象成资源（通过url来标识），资源有唯一id指向，每次对资源的访问会改变这些资源；响应中包含下一次访问要访问的资源id

REST & SOAP 

SOAP：给一个固定的方式供你调用

REST：更加灵活。服务器和客户机解耦（用涵盖在响应中的下一个资源的uri（id）），服务器可以不断修正自己的功能，对客户没有什么影响

HATEOAS超媒体应用状态引擎

使用HTTP状态码

给出错误描述信息

#### Mashup（混搭）

定义：混搭是一种web应用，从两个以上的源把数据集成到一个集成应用中

混搭原则：不混乱，有足够的内聚性】

**混搭**（[英语](https://zh.wikipedia.org/wiki/%E8%8B%B1%E8%AF%AD)：**Mashup**）指整合[网络](https://zh.wikipedia.org/wiki/%E7%B6%B2%E8%B7%AF)上多个资料来源或功能，以创造新服务的网络应用程序。

服务器端：PHP

客户端

如何：

抽取型混搭

流混搭

对象：消费级；企业级