Angular

简介

起源

- 2009年Google Feedback Project
- 将6个月开发的17000行前端代码，使用3周压缩到1500行

简介

- Angularjs 致力于减轻开发人员在开发AJAX应用过程中的痛苦
- 官网:http://www.angularjs.org/

概念

- 客户端模版
- MVC
- 数据绑定
- 依赖注入

客户端模版

- Angular中，模版和数据都会被发送到浏览器中，然后在客户端进行装配

MVC

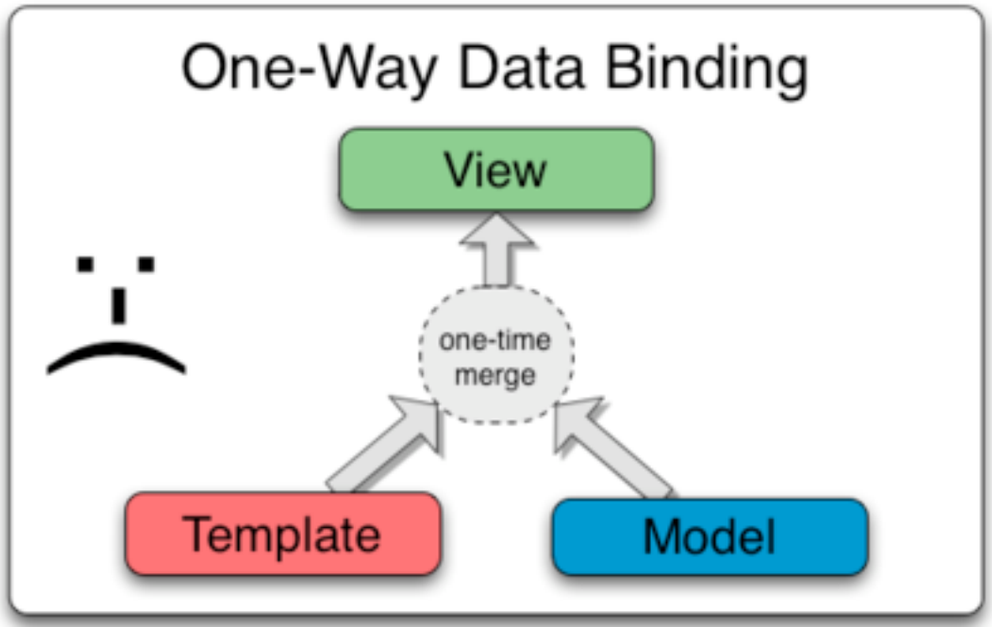
- MVC核心概念：把管理数据的代码(model)、应用逻辑代码(controller)、向用户展示数据的代码(view)清晰地分离开

- Angularjs应用中:
  视图就是Document Object Model
  控制器就是Javascript类
  模型数据则被存储在对象的属性中

数据绑定

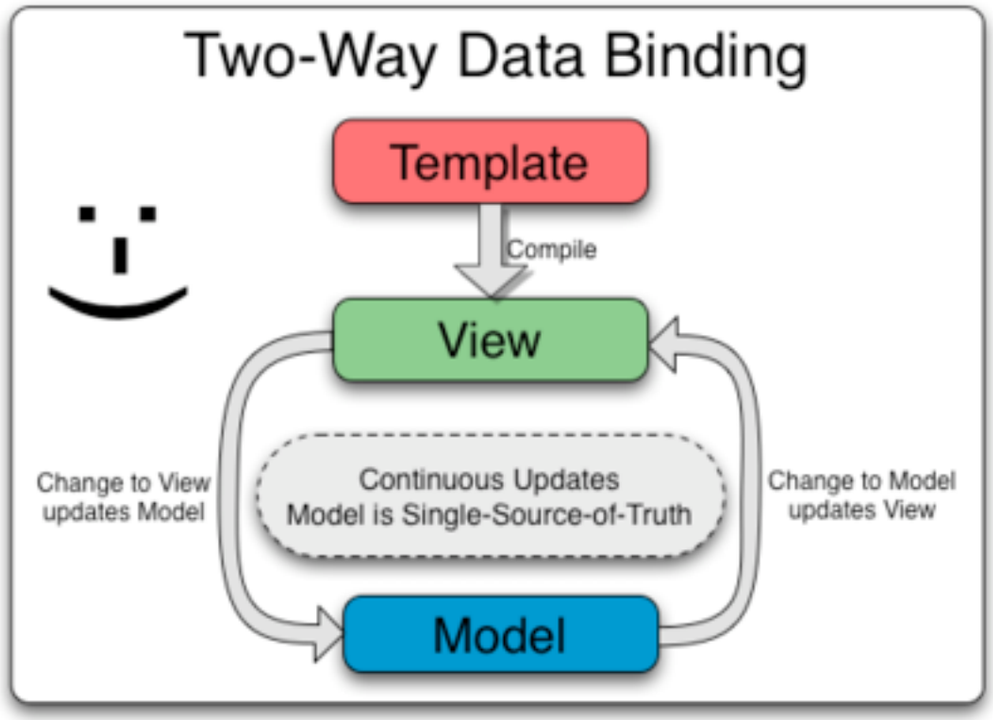
- 数据绑定可自动将model和view间的数据同步。
- Angular实现数据绑定的方式，可以让我们把model当作程序中唯一可信的数据来源。view始终是model的投影。当model发生变化时，会自动反映到view上。

经典模板系统中的数据绑定



- 大多数模板系统中的数据绑定都是单向的
- 把模板与model合并在一起变成view，如果在合并之后，model发生了变化，不会自动反映到view上。
- 用户在view上的交互也不会反映到model中，开发者必须写大量代码不断地在view与model之间同步数据。

Anguarjs 模板中的数据绑定



- 模板是在浏览器中编译的，在编译阶段产生了一个实时更新(live)的视图
- 不论在model或是view上发生了变化，都会立刻反映到对方。
- model成为程序中唯一真实的数据来源，极大地简化了开发者需要处理的编程模型。

依赖注入

- 依赖注入是一种软件设计模式，用来处理代码的依赖关系。

- Angular的依赖注入只是简单的获取它所需要的东西，而不需要创建那些他们所依赖的东西

ng-app

- ng-app指令告诉Angular应该管理页面中的哪一块

模版显示文本 && ng-bind

- {{expression}}
- <tag ng-bind="expression"></tag>

javascript 表达式

- var str = ‘alert(1+2)’;
- eval(str)

angular 表达式

- angular 表达式 通过$parse服务解析执行。

- 与Javascript 表达式的区别
  1.属性求值：所有属性的求值是对于scope的，而javascript是对于window对象的。
  2.宽容：表达式求值，对于undefined和null，angular是宽容的，但Javascript会产生NullPointerExceptions =
  3.没有流程控制语句：在angular表达式里，不能做以下任何的事：条件分支、循环、抛出异常
  4.过滤器（filters）：我们可以就将表达式的结果传入过滤器链（filter chains）

ng-controller

- 控制器就是你所编写的类或者类型，它的作用是告诉Angular该模型是由哪些对象或者基本数据结构构成的

$scope

什么是 scope

- scope是一个指向应用model的object，也是表达式的执行上下文。
- scope被放置于一个类似应用的DOM结构的层次结构中。

scope的特性

- scope提供$watch API,用于监测model的变化。
- scope提供$apply，在"Angular realm"（controller、server、angular event handler）之外，从系统到视图传播任何model的变化。
- scope可以在提供到被共享的model属性的访问的时候，被嵌入到独立的应用组件中。scope通过（原型），从parent scope中继承属性。

$apply

- $scope.$apply(expression)
- $apply()方法可以在angular框架之外执行angular JS的表达式，例如：DOM事件、setTimeout、XHR或其他第三方的库

angular是怎么知道变量发生了改变

- 要知道一个变量变了，方法不外乎两种

- 1.能通过固定的接口才能改变变量的值，比如说只能通过 set() 设置变量的值，set被调用时比较一下就知道了。这中方法的缺点洗是写法繁琐

- 2.脏检查，将原对象复制一份快照，在某个时间，比较现在对象与快照的值，如果不一样就表明发生变化，这个策略要保留两份变量，而且要遍历对象，比较每个属性，这样会有一定性能问题

angular的策略

- angular的实现是使用脏检查

- angular的策略
  1.不会脏检查所有的对象，当对象被绑定到html中，这个对象添加为检查对象（watcher）。
  2.不会脏检查所有的属性，同样当属性被绑定后，这个属性会被列为检查的属性。

- 在angular程序初始化时，会将绑定的对象的属性添加为监听对象（watcher），也就是说一个对象绑定了N个属性，就会添加N个watcher。

什么时候去脏检查

- angular 所系统的方法中都会触发比较事件，比如：controller 初始化的时候，所有以ng-开头的事件执行后，都会触发脏检查

手动触发脏检查

- $apply仅仅只是进入angular context ,然后通过$digest去触发脏检查
- $apply如果不给参数的话，会检查该$scope里的所有监听的属性，推荐给上参数

$digest()

- 所属的scope和其所有子scope的脏检查，脏检查又会触发$watch()，整个Angular双向绑定机制就活了起来~

- 不建议直接调用$digest()，而应该使用$apply()，$apply其实不能把信直接送给$digest，之间还有$eval门卫把关，如果$apply带的表达式不合法，$eval会把错误送交$exceptionHandler，合法才触发digest，所以更安全

$watch

- 在digest执行时，如果watch观察的value与上次执行时不一样时，就会被触发

- AngularJS内部的watch实现了页面随model的及时更新

- $watch(watchFn,watchAction,deepWatch)
  watchFn:angular表达式或函数的字符串
  watchAction(newValue,oldValue,scope):watchFn发生变化会被调用
  deepWacth:可选的布尔值命令检查被监控的对象的每个属性是否发生变化

- $watch会返回一个函数，想要注销这个watch可以使用函数

scope生命周期

- 1.用户请求应用起始页
- 2.angular 被加载，查找ng-app指令
- 3.Angular 遍历模版，查找指令
- 4.controller被启用，$scope被注入进来
- 5.在模版link过程中，指令在scope中注册$watch。这些watch将会被用作向DOM传播model的值。
- 5.当在controller中做同步的工作时angular API 已经隐式地做了$apply操作
- 6.在$apply的结尾，angular会在root scope执行一个$digest周期，这将会传播到所有child scope中。在$digest周期中，所有注册了$watch的表达式或者function都会被检查，判断model是否发生了改变，如果改变发生了，那么对应的$watch监听器将会被调用。
- 7.当child scope不再是必须的时候，child scope的产生者有责任通过scope.$destroy() API销毁它们（child scope）。这将会停止$digest的调用传播传播到child scope中，让被child scope model使用的内存可以被gc回收。

$eval && $evalAsync

- 在作用域的上下文中执行表达式
- $eval(expression)
- $evalAsync接受一个函数，把它列入计划，在当前正持续的digest中或者下一次digest之前执行，即使它已经被延迟了，仍然会在现有的digest遍历中被执行,类似于settimeout(fun,0)

$broadcast && $emit && $on

- $broadcast:会把事件广播给所有子controller
- $emit:则会将事件冒泡传递给父controller
- $on:是angular的事件注册函数

- 可以简单实现解决angular controller之间的通信
- 事件也会产生一个event对象，类似与dom中的事件冒泡等机制

$new && $destroy

- $new 创建一个新的作用域
- $destory 销毁一个作用域
- 作用域的继承是类似于javascript的原型继承，会有类似的原型链

Module

什么是Module

- 大部分应用都有一个主方法(main)用来实例化、组织、启动应用。
- AngularJS应用没有主方法，而是使用模块来声明应用应该如何启动。
- 模块允许通过声明的方式来描述应用中的依赖关系，以及如何进行组装和启动

Angular 模块

- 模块是组织业务的一个框框，在一个模块当中定义多个服务。当引入了一个模块的时候，就可以使用这个模块提供的一种或多种服务了。

- AngularJS 本身的一个默认模块叫做 ng ， 它提供了 $http ， $scope 等等服务

- 服务只是模块提供的多种机制中的一种，其它的还有指令（ directive ），过滤器（ filter ），及其它配置信息。

- 也可以在已有的模块中新定义一个服务，也可以先新定义一个模块，然后在新模块中定义新服务。

- 服务是需要显式地的声明依赖（引入）关系的，让 ng 自动地做注入

Module优点

- 启动过程是声明式的，更容易懂。
- 在单元测试是不需要加载全部模块的，因此这种方式有助于写单元测试。
- 可以在特定情况的测试中增加额外的模块，这些模块能更改配置，能帮助进行端对端的测试。
- 第三方代码可以作为可复用的module打包到angular中
- 模块可以以任何先后或者并行的顺序加载（因为模块的执行本身是延迟的）。

ng-app

- 通过ng-app指定对应的模块应用启动

定义模块

- angular.module(name[, requires], configFn);
- configFn 会在模块初始化时执行，可以在里配置模块的服务
- configFn @see angular.config()

定义服务 $provider

- 服务本身是一个任意的对象。
- ng 提供服务的过程涉及它的依赖注入机制。
- angular 是用$provider对象来实现自动依赖注入机制，注入机制通过调用一个 provider 的 $get() 方法，把得到的对象作为参数进行相关调用
- $provider.provider 是一种定义服务的方法，$provider还提供了很多很简便的方法,这些简便的方法还直接被module所引用

$provider.factory

- factory 方法直接把一个函数当成是一个对象的 $get() 方法
- @see module.factory
- 返回的内容可以是任何类型

$provider.service

- 和factory类似，但返回的东西必须是对象
- @see module.service

显式和隐式依赖注入

- 把service当作被依赖的资源加载到controller中的方法，与加载到其他服务中的方法很相似。

- javascript是一个动态语言，DI不能弄明白应该通过参数类型注入哪一个service

- 显式依赖:要通过$inject属性指定service名称， 它是一个包含需要注入的service名称的字符串数组，工厂方法中的参数顺序，与service 在数组中的顺序一致。

- 隐式依赖:则允许通过参数名称决定依赖，$scope

Filters

什么是angular 过滤器

- 是用于对数据的格式化，或者筛选的函数,可以直接在模板中通过一种语法使用
- {{ expression | filter }}
- {{ expression | filter1 | filter2 }}
- {{ expression | filter1:param,….}}

过滤器种类

- number
- currency
- date
- limitTo
- lowercase
- uppercase
- filter
- json
- orderBy

自定义过滤器

- module.filter(name, filterFactory)
- @$filterProvider.register().

Controllers

angular controller

- 在angular中，controller是一个javascript 函数（type/class），被用作扩展除了root scope在外的angular scope的实例。

- 也可以通过module.controller(name, constructor)

- @see $controllerProvider.register().

- controller可以用作:
  设置scope对象的初始状态。
  增加行为到scope中。

正确的使用controller

- controller不应该尝试做太多的事情。它应该仅仅包含单个视图所需要的业务逻辑
- 保持Controller的简单性，常见办法是抽出那些不属于controller的工作到service中，在controller通过依赖注入来使用这些service
- 不要在Controller中做以下的事情：
  1.任何类型的DOM操作 - controller应该仅仅包含业务逻辑，任何表现逻辑放到controller中，大大地影响了应用逻辑的可测试性。angular为了自动操作（更新）DOM，提供的数据绑定。如果希望执行我们自定义的DOM操作，可以把表现逻辑抽取到directive中。
- 2.Input formatting（输入格式化） - 使用angular form controls 代替。
- 3.Output filtering （输出格式化过滤） - 使用angular filters 代替。
- 4.执行无状态或有状态的、controller共享的代码 - 使用angular services 代替。
- 5.实例化或者管理其他组件的生命周期（例如创建一个服务实例）。

Directive

什么是指令

- 可以利用指令来扩展HTML标签，增加声明式语法来实现想做的任何事，可以对应用有特殊意义的元素和属性来替换一般的HTML标签

- angular也内置了非常多的指令，ng-app、ng-controller

指令和HTML校验

- angular 内置指令的语法，已ng开始，代表angular命名空间，连接符后面的内容代表指令的名称
- 指令的语法在很多HTML校验规则中是不合法的，Angular提供了多种调用指令方法，可以顺利通过不同校验的规则

校验器	格式	示例
none	namespace-name	ng-bind
XML	namespace:name	ng:bind
HTML5	data-namespace-name	data-ng-bind
XHTML	x-namespace-name	x-ng-bind

指令的执行过程

- 浏览器得到 HTML 字符串内容，解析得到 DOM 结构。
- ng 引入，把 DOM 结构扔给 $compile 函数处理：
- 找出 DOM 结构中有变量占位符
- 匹配找出 DOM 中包含的所有指令引用
- 把指令关联到 DOM
- 关联到 DOM 的多个指令按权重排列
- 执行指令中的 compile 函数（改变 DOM 结构，返回 link 函数）
- 得到的所有 link 函数组成一个列表作为 $compile 函数的返回
- 执行 link 函数（连接模板的 scope）。

Angular 内置指令

渲染指令

- ng-init
- ng-bind
- ng-repeat

  $index 当前索引
  $first 是否为头元素
  $middle 是否为非头非尾元素
  $last 是否为尾元素

- ng-include
- ng-bind-template

事件指令

- ng-change
- ng-click
- ng-dblclick
- ng-mousedown
- ng-mouseenter
- ng-mouseleave
- ng-mousemove
- ng-mouseover
- ng-mouseup
- ng-submit

节点指令

- ng-style
- ng-class
- ng-class-even
- ng-class-odd
- ng-show
- ng-hide
- ng-switch
- ng-src
- ng-href
- ng-if

Angular 自定义指令

指令的定义

- module.directive(name, directiveFactory)
- @see $compileProvider.directive()

指令的名字

- 请不要使用ng为指令命名，这样可能会和angular内置指令冲突
- 如果指令的名字为xxx-yyy 在设置指令的名字时应为xxxYyy 驼峰式声明法

指令定义选项

- priority
- terminal
- scope
- controller
- controllerAs
- require
- restrict
- template
- templateUrl
- replace
- transclude
- compile
- link

restrict

- restrict:指令在模版中的使用方式
- 可以4种风格任意组合，如果忽略restrict，默认为A
- 如果打算支持IE8，请使用基于属性和样式类的指令

字母	风格	示例
E	元素	<my-dir></my-dir>
C	样式类	
A	属性	
M	注释	<!-- directive: my-dir exp -->

template

- template:模板内容，这个内容会根据 replace 参数的设置替换节点或只替换节点内容。

replace

- replace:如果此配置为true则替换指令所在的元素，如果为false或者不指定，则把当前指令追加到所在的元素内部

- 对于restrict为元素(E)在最终效果中是多余的，所有replace通常设置为true

templateUrl

- templateUrl:加载模版所要使用的URL
- 可以加载当前模板内对应的的text/ng-template script id
- 在使用chrome浏览器时，"同源策略"会阻止chrome从file://中加载模版，并显示一个"Access-Control-Allow-Origin" 不允许源为null，可以把项目放在服务器上加载，或者给Chrome设置一个标志，命令为:
chrome —allow-file-access-from-files

transclude

- transclude:指令元素中的原来的子节点移动到一个新模版内部
- 当为true时，指令会删掉原来的内容，使你的模版可以用ng-transclude指令进行重新插入

priority && terminal

- priority:设置指令在模版中的执行顺序，顺序是相对于元素上其他执行而言，默认为0，从大到小的顺序依次执行

- 设置优先级的情况比较少，象ng-repeat,在遍历元素的过程中，需要angular先拷贝生成的模版元素，在应用其他指令，所以ng-repeat默认的priority是1000

- terminal 是否以当前指令的权重为结束界限。如果这值设置为 true ，则节点中权重小于当前指令的其它指令不会被执行。相同权重的会执行。

Angularjs 指令编译三阶段

- 1. 标准浏览器API转化
  将html转化成dom，所以自定义的html标签必须符合html的格式

- 2. Angular compile
  搜索匹配directive，按照priority排序，并执行directive上的compile方法

- 3. Angular link
  执行directive上的link方法，进行scope绑定及事件绑定

为什么编译的过程要分成compile和link?

- 简单的说就是为了解决性能问题，特别是那种model变化会影响dom结构变化的，而变化的结构还会有新的scope绑定及事件绑定，比如ng-repeat

compile和link的使用时机

- compile
  想在dom渲染前对它进行变形，并且不需要scope参数
  想在所有相同directive里共享某些方法，这时应该定义在compile里，性能会比较好
  返回值就是link的function，这时就是共同使用的时候

- link
  对特定的元素注册事件
  需要用到scope参数来实现dom元素的一些行为

compile

- complie:function(tElement,tAttrs,transclude)
- complie函数用来对模版自身进行转换，仅仅在编译阶段运行一次
- complie中直接返回的函数是postLink，表示link参数需要执行的函数，也可以返回一个对象里面包含preLink和postLink
- 当定义complie参数时，将无视link参数，因为complie里返回的就是该指令需要执行的link函数

link

- link(scope,iElement,iAttrs,controller)
- link参数代表的是complie返回的postLink
- preLink 表示在编译阶段之后，指令连接到子元素之前运行
- postLink 表示会在所有子元素指令都连接之后才运行
- link函数负责在模型和视图之间进行动态关联，对于每个指令的每个实例，link函数都会执行一次

controller && controllerAs && require

- controller 他会暴露一个API，利用这个API可以在多个指令之间通过依赖注入进行通信
- controller($scope,$element,$attrs,$transclude)
- controllerAs 是给controller起个别名，方便使用
- require 可以将其他指令传递给自己

选项	用法
directiveName	通过驼峰法的命名指定了控制器应该带有哪一条指令,默认会从同一个元素上的指令
^directiveName	在父级查找指令
?directiveName	表示指令是可选的，如果找不到，不需要抛出异常

scope

- scope:为当前指令创建一个新的作用域，而不是使之继承父作用域

- false 继承父元素的作用域
  true 创建一个新的作用域
  object 独立的scope

- object:参数
  &:作用域把父作用域的属性包装成一个函数，从而以函数的方式读写父作用域的属性
  =:作用域的属性与父作用域的属性进行双向绑定，任何一方的修改均影响到对方
  @:只能读取父作用域里的值单项绑定

Module里的一些其他方法

constant

- constant(name,object)
- 此方法首先运行，可以用它来声明整个应用范围内的常量，并且让它们在所有配置(config方法里)和实例(controller,service等)方法中都可用

value

- value(name,object)
- 如果只想在服务内得到一些内容，可以通过value来申明常量

run

- run(initializationFn)
- 想要在注入启动之后执行某些操作，而这些操作需要在页面对用户可用之前执行，可以使用此方法
- 比如加载远程的模版，需要在使用前放入缓存，或者在使用操作前判断用户是否登录，未登录可以先去登陆页面

Form

表单

- 一般来讲表单可能遇到的问题
  1.如何数据绑定
  2.验证表单
  3.显示出错信息
  4.整个Form的验证
  5.避免提交没有验证通过的表单
  7.防止多次提交

input type 扩展

- number
- url
- email
- reset

input 属性

- name 名字
- ng-model 绑定的数据
- ng-required 是否必填
- ng-minlength 最小长度
- ng-maxlength 最大长度
- ng-pattern 匹配模式
- ng-change 值变化时的回调

CSS样式

- ng-valid 当表单验证通过时的设置
- ng-invalid 表单验证失败时的设置
- ng-pristine 表单的未被动之前拥有
- ng-dirty 表单被动过之后拥有

Form控制变量

- 字段是否未更改
  formName.inputFieldName.$pristine
- 字段是否更改
  formName.inputFieldName.$dirty
- 字段有效
  formName.inputFieldName.$valid
- 字段无效
  formName.inputFieldName.$invalid
- 字段错误信息
  formName.inputfieldName.$error

From 方法

- $setPristine 将表单复位原始状态，class,$dirty,$pristine

ng-model

- ng-model是angular原生的directive
- 可以通过require ngModel 可以更深入的去处理数据的双向绑定

ngModel 里的属性

- $parsers属性 保存了从viewValue向modelValue绑定过程中的处理函数，它们将来会依次执行
- $formatters 它保存的是从modelValue向viewValue绑定过程中的处理函数
- $setViewValue 当view发生了某件事情时，从view向model绑定调用$setViewValue 把viewValue保存下来
- $render 当模型发生变化时，应该怎么去更新视图，从model向view绑定，调用ctrl.$render方法，将viewValue渲染到页面上
- $setValidity 设置验证结果
- $viewValue 视图的值
- $modelValue 模型里的值

XHR和服务器端的通信

$http

- $http 是一个服务，简单的封装了XMLHttpRequest对象
- $http(config).success(fun).error(fun)

$http短名方法

- $http.get()
- $http.delete()
- $http.header()
- $http.jsonp()
- $http.post()
- $http.put()

$http 配置对象

- method
- url
- params
- data
- headers
- xsrfHeaderName
- xsrfCookieName
- transformRequest
- transformResponse
- cache
- withCredentials
- timeout
- responseType

responseType

- " (string – default)
- "arraybuffer" (ArrayBuffer)
- "blob" (blob object)
- "document" (HTTP document)
- "json" (JSON object parsed from a JSON string)
- "text" (string)
- "moz-blob" (Firefox to receive progress events)
- "moz-chunked-text" (streaming text)
- "moz-chunked-arraybuffer" (streaming ArrayBuffer)