Chrome内部的scheme管理

Chrome 和 CEF都会在创建net::URLRequestContextGetter(在chrome中由ChromeURLRequestContextGetter实现, 在CEF中由CefURLRequestContextGetter实现)的时候会创建并填充一个map<scheme, protocal\_handler>, 通过net::URLRequestJobFactoryImpl::SetProtocalHandler(scheme, protocal\_handler)设置给net::URLRequestJobFactory, (SetProtocolHanlder不是URLRequestJobFactory的方法, 所以Chrome和CEF都会在net之外的content中包含impl头文件, 创建impl实例). content api没有提供接口让使用者定制scheme的注册(哪些scheme是合法的, 标准的)和处理(解析scheme的东西),我们需要像chrome一样在创建net::URLRequestContextGetter的时候填充一些内置的protocolHandler, 也需要像CEF一样提供接口注册scheme和对应的处理对象.

Content API 指Content API使用者可以使用的接口

CEF API 指CEF API使用者可以调用的接口, Cef 对Content API的调用使用Content API的颜色, 意味着咱们可以有样学样地调用

Chrome Impl 指Chrome的实现, Chrome对Content API的使用也会使用Content API的颜色

* 概述:

对于哪些Scheme是合法的, Cef和Chrome都有调用ChildProcessSecurityPolicy;

对于定制scheme的解析方法, 二者都有调用content::URLRequestJobFactoryImpl::SetProtocolHandler()来定制某些scheme的处理,定制的ProtocalHandler根据scheme来创建具体的URLRequestJob. 对于内置的scheme, 通过ProtocolFactory来处理请求.

Chrome的scheme处理分为内置(kBuiltinFactories处理的), Chrome定制的(调用URLRequestJobFactoryImpl::SetProtocalHandler加上的).

[遗留问题]还看到一个接口content::WebContentsDelegate::RegisterProtocolHandler()和WebCore::Navigator::RegisterProtocolHandler()应该是可以定制scheme处理方法的, 但是还没有弄明白怎么用.

* CEF 怎么做到定制scheme的处理的:

CEF的注册接口bool AddCustomScheme(const CefString& scheme\_name, bool is\_standard, bool is\_local, bool is\_display\_isolated) 会做的事情:

1. 在CefContentClient里会保存一份合法scheme的信息. 通过ChildProcessSecurityPolicy来指定合法的webSafeScheme,
2. CefContentClient::AddAdditionalSchemes()被content调用的时候返回通过CefSchemeRegistrar::AddCustomScheme()加上的standard schemes,

ContentClient::AddAdditionalSchemes()的注释为”Gives the embedder a chance to register its own standard and saveable url schemes early on in the startup sequence.”.

注册之后, Cef通过bool CefRegisterSchemeHandlerFactory( const CefString& scheme\_name, const CefString& domain\_name,

CefRefPtr<CefSchemeHandlerFactory> factory); 来指定解析特定scheme的实现, CefSchemeHandlerFactory提供CefResourceHandler, 这是一组由使用者定制的处理request的接口(process和get response等). 简单来讲, Cef通过调用net::URLRequestJobFactoryImpl::SetProtocalHandler(scheme, ProtocalHandler)来实现, Cef实现net::URLRequestJobFactory::ProtocolHandler::MayCreateJob()来根据request创建Job.

Content没有提供统一的Scheme管理类, 列表, map之类的, Cef的CefContentClient:: scheme\_info\_list\_会保存scheme info.

Cef的实现scheme:: AddInternalSchemes会在CefContentClient::AddAdditionalSchemes()被调用的时候加两个内置scheme(会加上chrome::kChromeUIScheme和chrome::kChromeDevToolsScheme)

* Cef可能需要注意的地方:

Custom scheme shoudl be registered in both browser/sub-processes. 浏览器和渲染进程中都需要注册custom scheme. 论坛上看来的准确性并不确定.

ChildProcessSecurityPolicy是全局单例.

Cef的AddCustomScheme()和CefRegisterSchemeHandlerFactory都需要使用才能够定制scheme的处理, 前者只是挂个名, 后者做实际处理.

* [注] content:: RegisterContentSchemes(bool lock\_standard\_schemes)

这个方法的注释有两个意思, 不要使用或重写这个接口虽然它被开放出来了, 要定制使用ContentClient::AddAdditionalSchemes.

* Chrome怎么做到scheme的定制处理的

和Cef一样, Chrome通过实现Content::ContentClient::AddAdditionalSchemes(standard\_schemes, savable\_schemes)来告诉content哪些scheme是”标准”的, Chrome的”标准”scheme统一保存在/src/googleurl/src/url\_util.cc中管理, 这是一些命名空间内的全局变量和方法, “savable scheme”放在savable\_url\_schemes.cc中保存. 后面这两个都不是content api的使用者需要关心的.

content层的使用者可以使用的scheme名字在src/content/public/common/url\_constants.h里.

下面是Chrome是怎样调用content::URLRequestJobFactoryImpl::SetProtocolHandler的:

在创建net::URLRequestContextGetter的过程中, content会在content::StoragePartitionImplMap::Get()中创建并填充一个ProtocalHandlerMap,Get()内部会往这个ProtocalHandlerMap里填上”blob”, “filesystem”和”chrome”和”chrome\_devtools”4个scheme和protocalhandler, 然后content给ContentBrowserClient一个机会通过GetAdditionalWebUISchemes()告诉content有哪些web uischeme, Chrome在这里传入了”chrome-search”, content(通过URLDataManagerBackend::CreateProtocolHandler)为它创建了一个protocal handler, 这五个schemes(blob, filesystem, chrome, chrome\_search, chrome-devtools)被传给ChromeURLRequestContextGetter的创建方法, ChromeURLRequestContextGetter在创建ChromeURLRequestContext的过程中又通过profile\_io\_data.cc(也是创建者传进来的, 根据profile生成scheme)添加了file, chrome-extension, chrome-extension-resource, data, about, ftp共6个scheme, 把这11个protocal handler封到protocol\_handler\_interceptor里作为处理request的链.

* 注入ProtocalHandler之后定制的Scheme处理方法被调用的过程:

我们已经知道content::ResourceLoader会使用net::URLRequestJobManager根据net::URLRequest来创建一个net::URLRequestJob(见url request部分的”浏览器对URLRequest的处理”), 在URLRequestJob\* URLRequestJobManager::CreateJob(request)的实现中我们看到ProtocolHandlerRegistry::JobInterceptorFactory:: MaybeCreateJobWithProtocolHandler(scheme, request)会首先得到一次机会使用注入的ProtocalHandler创建job, 情况1: 在Chrome浏览器中如果url是chrome之类的会通过ProtocalHandler创建job; 情况2: 如果是http, https, file, ftp, about, data这些kBuiltinFactories中的一个, 会调用net提供的对应的ProtocolFactory来创建Job. (kBuiltinFactories和Chrome注入的protocal handler里都有file, protocal handler会先于built in factories处理file scheme, 由FileProtocalHandler返回一个URLRequestFileJob).

* Chrome通过profile确定注入哪些scheme的protocol handler

[遗留问题]/chrome/browser/profiles/ProfileIOData会读取磁盘上AppData的数据并据此创建一些scheme和protocol handler, 此条待验证.

* Content内部实现了这些ProtocalHandler: DevToolsJobFactory, BlobProtocolHandler

net模块提供了这些: DataProtocolHandler, FileProtocolHandler, FtpProtocolHandler,

Cef和Chrome都提供了: ChromeProtocolHandler