**DCOS应用市场之应用仓库的管理**

上期，我们介绍了DC/OS应用市场里面的开源项目universe,并且介绍了universe的应用文件内容和创建过程，当用户创建了属于自己的universe repository后，如何使用这些repository呢？

这期，我们从开始介绍DC/OS的cosmos来逐层抛析应用仓库的管理。在DC/OS系统中，应用仓库管理的后端是通过cosmos提供的服务完成的。

首先，我们先简单介绍一下cosmos。

**Cosmos简介**

Cosmos是DCOS的一个开源项目（github地址在<https://github.com/dcos/cosmos>），t通过对外提供REST API，完成了对universe仓库的管理以及应用服务的安装部署，在DC/OS系统中，cosmos是通过systemd管理起来的系统服务，用户可以通过系统服务管理命令systemctl来控制cosmos的启停。

下面是Cosmos的系统启动文件，DCOS的系统安装完成后，默认放在/etc/systemd/system/dcos-cosmos.service下面:

[Unit]

Description=Package Service: DC/OS Packaging API

After=dcos-mesos-master.service

After=dcos-gen-resolvconf.service

[Service]

Restart=always

StartLimitInterval=0

RestartSec=15

EnvironmentFile=/opt/mesosphere/environment

EnvironmentFile=-/var/lib/dcos/environment.proxy

ExecStartPre=/bin/ping -c1 ready.spartan

ExecStartPre=/opt/mesosphere/bin/exhibitor\_wait.py

## CoreOS

ExecStartPre=-/usr/bin/mkdir -p /var/lib/cosmos

## Ubuntu

ExecStartPre=-/bin/mkdir -p /var/lib/cosmos

ExecStart=/opt/mesosphere/bin/java -Xmx2G -classpath /opt/mesosphere/packages/cosmos--25d98ad8c31c73550a40c8e1022c08f2e53976c4/lib/:/opt/mesosphere/packages/cosmos--25d98ad8c31c73550a40c8e1022c08f2e53976c4/usr/cosmos.jar com.simontuffs.onejar.Boot

**Cosmos服务的启动**

Cosmos的核心代码是用scala实现的，通过cosmos在启动后对外提供了REST API, DCOS的UI通过调用Cosmos开放出来的REST API来完成相应的工作。

Cosmos通过twitter 开源出来的finagle，来完成创建了一个http server, 在服务启动过程中，分别注册了adminrouter, marathon,mesos以及zookeeper这四大家族的client,其中，其中adminrouter是dcos的开源项目之一（https://github.com/dcos/adminrouter）想要更细一些了解的同学可以参考github，其目的就是做了反向代理，能client通过一个url + port的方式访问到底层的各种服务；那marathon的client上期我们大概介绍过，是DC/OS的服务都是托管在Marathon上面，所以这里需要一个marathon的client来通过调用marathon相应的API来完成应用的创建；mesos上期也大概提了下，在DC/OS的应用市场里，除了一部分能跑在marathon上的应用外，还有一些应用是需要跑在mesos上来作为mesos的framework的，所以没有client不行啊，剩下的zookeeper就是用来存放用户在dcos ui上添加的universe的repository的。

下面看看启动参数：

Cosmos编译完是个jar包，启动的时候你的环境需要有java，目前后面可以配置的参数有：

1. dcosUri: 系统启动时可以通过指定 “-com.mesosphere.cosmos.dcosUri=<>”来告诉cosmos你的dcos的地址在哪。
2. adminRouterUri: 系统启动时可以通过指定 “-com.mesosphere.cosmos.adminRouterUri=<>”来指定adminrouter的地址
3. marathonUri: 系统启动时可以通过指定 “-com.mesosphere.cosmos.marathonUri=<>“ 来指定marathon的地址
4. mesosMasterUri: 系统启动时可以通过指定 “-com.mesosphere.cosmos.mesosMasterUri =<>” 来指定你的mesos master的地址。
5. zookeeperUri: 系统启动时可以通过指定 “-com.mesosphere.cosmos.zookeeperUri =<>”
6. dataDir: 系统启动时可以通过指定 “-com.mesosphere.cosmos. dataDir =<>”来指定一个work目录，其目的是用来存放universe repository的那些打包的应用的，下面会详细介绍。

好了，能指定的参数都在这里了，在写这个的时候，不小心更新了下code（其实好久都没有更新过了）,看到社区又加了两个环境变量进来了，这里顺带介绍一下：

这两个环境变量是和zookeeper的authentication有关的，分别是ZOOKEEPER\_USER和ZOOKEEPER\_SECRET。当然，你export了cosmos肯定会认得，不信摘一点Cosmos的代码让你看看：

val boot = ar map { adminRouter =>

val zkUri = zookeeperUri()

logger.info("Using {} for the ZooKeeper connection", zkUri)

val marathonPackageRunner = new MarathonPackageRunner(adminRouter)

val zkClient = zookeeper.Clients.createAndInitialize(

zkUri, sys.env.get("ZOOKEEPER\_USER").zip(sys.env.get("ZOOKEEPER\_SECRET")).headOption

)

onExit {

zkClient.close()

}

好了，说了这么多，总算能启动服务了吧：

root@mesos8:~# java -Xmx2G -classpath /root/hchen/cosmos-server\_2.11-0.1.6-SNAPSHOT-one-jar.jar com.simontuffs.onejar.Boot -com.mesosphere.cosmos.dcosUri=http://9.111.255.40 -com.mesosphere.cosmos.zookeeperUri=zk://127.0.0.1:2181/cosmos -com.mesosphere.cosmos.kubernetesUri=http://0.0.0.0:8888 -com.mesosphere.cosmos.dataDir=/var/lib/cosmos

我平常就这么启动的，当然，你可以通过systemctl dcos-cosmos start/stop来控制cosmos服务的启停。

好了，服务也起来了，下面就看看使用了。

**仓库添加**

在DC/OS的GUI上面，用户可以在system -> Overview -> Repositories页面中查询，添加及删除universe 的repositories.

我们先了解一下仓库的添加。

DCOS系统安装好后，默认会有一个mesosphere官方提供的default repository, cosmos服务在启动时，通过读取zookeeper里面的数据来决定是否需要添加default repository, 当然，刚安装好的环境里面zookeeper里面肯定没有东西，所以cosmos服务在第一次启动的时候，default repository就会被添加进去。

private[this] val DefaultRepos: List[PackageRepository] = DefaultRepositories().getOrElse(Nil)

override def read(): Future[List[PackageRepository]] = {

Stat.timeFuture(stats.stat("read")) {

readFromZooKeeper.flatMap {

case Some((\_, bytes)) =>

Future(decodeData(bytes))

case None =>

create(DefaultRepos)

}

}

}

其中，default repository的文件是hard code的一个default-repositories.json文件，文件默认放在cosmos的lib目录下:

[root@hcmesos3 ~]#cat /opt/mesosphere/packages/cosmos--25d98ad8c31c73550a40c8e1022c08f2e53976c4/lib/default-repositories.json

[

{

"name": "Universe-1.7",

"uri": "https://universe.mesosphere.com/repo-1.7"

},

{

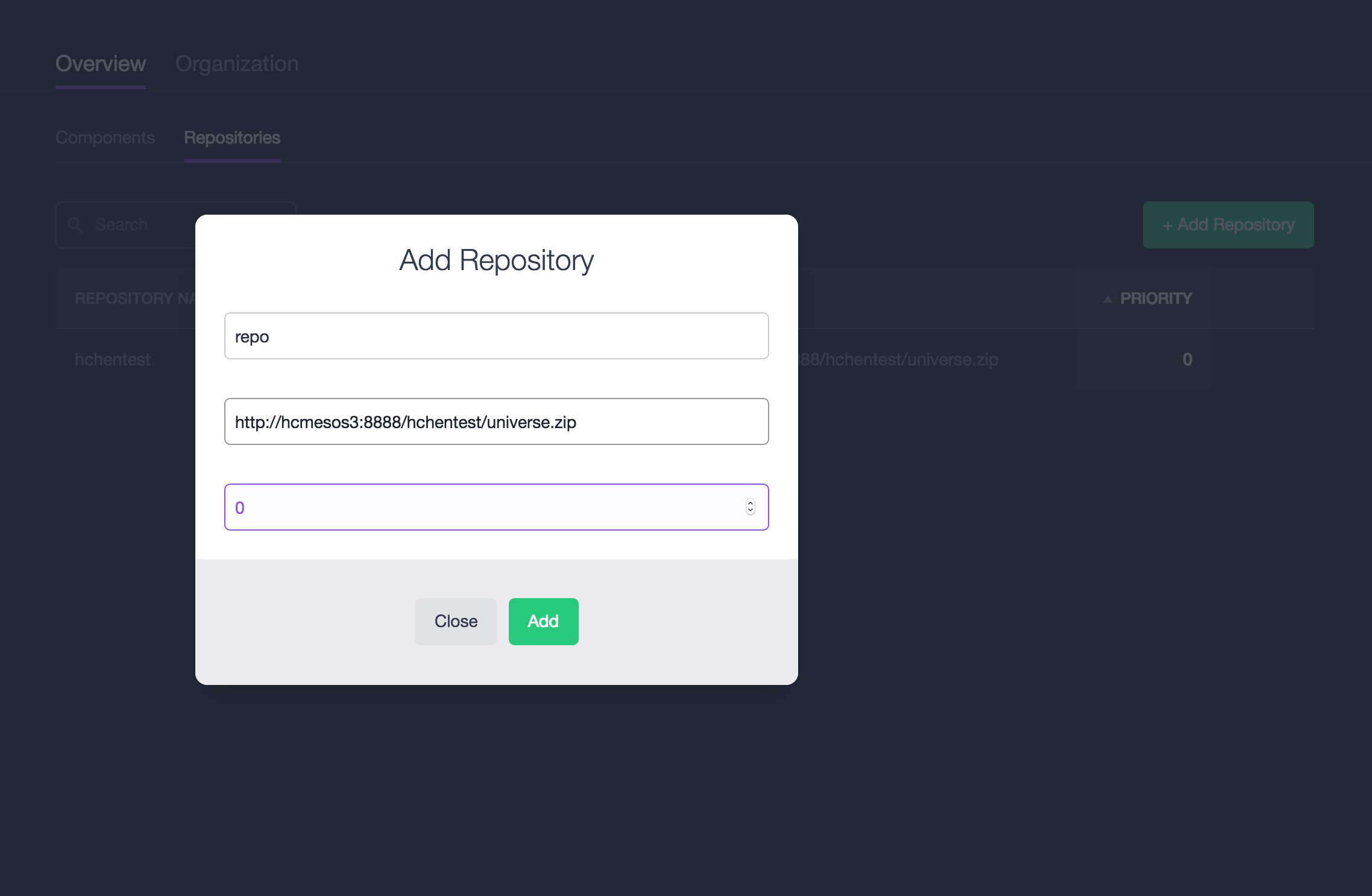
"name": "Universe",

"uri": "https://universe.mesosphere.com/repo"

}

]

当然，用户也可以添加自己的universe repository.



当用户进入页面开始添加一个universe的repositories的时候，universe会去调用CosmosPackageAction.js::addRepository来开始添加一个repository.

addRepository: function (name, uri, index) {

RequestUtil.json({

contentType: getContentType('repository.add', 'request', 'v1'),

headers: {Accept: getContentType('repository.add', 'response', 'v1')},

method: 'POST',

url: `${Config.rootUrl}${Config.cosmosAPIPrefix}/repository/add`,

data: JSON.stringify({name, uri, index}),

timeout: REQUEST\_TIMEOUT,

success: function (response) {

AppDispatcher.handleServerAction({

type: REQUEST\_COSMOS\_REPOSITORY\_ADD\_SUCCESS,

data: response,

name,

uri

});

},

error: function (xhr) {

AppDispatcher.handleServerAction({

type: REQUEST\_COSMOS\_REPOSITORY\_ADD\_ERROR,

data: RequestUtil.getErrorFromXHR(xhr),

name,

uri

});

}

});

},

前端dcos UI的请求会以POST的方式发送给 cosmos受理repository/add的请求，并且等待cosmos的response. 根据universe请求的URL, Cosmos会调用PackageRepositoryAddHandler::sourceStorage.add来完成操作，如下所示：

private[cosmos] final class PackageRepositoryAddHandler(

sourcesStorage: PackageSourcesStorage

) extends EndpointHandler[PackageRepositoryAddRequest, PackageRepositoryAddResponse] {

override def apply(request: PackageRepositoryAddRequest)(implicit

session: RequestSession

): Future[PackageRepositoryAddResponse] = {

request.uri.scheme match {

case Some("http") | Some("https") =>

sourcesStorage.add(

request.index,

PackageRepository(request.name, request.uri)

) map { sources =>

PackageRepositoryAddResponse(sources)

}

case \_ => throw UnsupportedRepositoryUri(request.uri)

}

}

其中sourcesStorage是个zooKeeperStorage的对象，添加的repositories通过ZooKeeperStorage对数据进行加密然后放到zookeeper上面（默认在zookeeper的/cosmos/package下面），下面是存储在zookeeper里面repository的信息。

WatchedEvent state:SyncConnected type:None path:null

[zk: localhost:2181(CONNECTED) 0] get /cosmos/package/repositories

{"metadata":{"Content-Type":"application/vnd.dcos.package.repository.repo-list+json;charset=utf-8;version=v1"},"data":"W3sibmFtZSI6ImhjaGVudGVzdCIsInVyaSI6Imh0dHA6Ly9tZXNvczEuZW5nLnBsYXRmb3JtbGFiLmlibS5jb20vYXBwX3N0b3JlL2N3Yy1hcHBzdG9yZS56aXAifV0="}

cZxid = 0x55

ctime = Thu Jun 02 04:25:49 EDT 2016

mZxid = 0x300

mtime = Sun Jul 10 23:01:31 EDT 2016

pZxid = 0x55

cversion = 0

dataVersion = 51

aclVersion = 0

ephemeralOwner = 0x0

dataLength = 249

numChildren = 0

顺便提一下，从上面的代码我们不难发现，目前universe的repository支持http和https两种协议。

除了对repository name和URL进行加密并且写到zookeeper里面以外，还会获取repository所提供的压缩包并且解压放到/var/lib/cosmos下(也就是启动的时候指定的dataDir)，里面就解压后的package index以及各种应用软件包的描述文件，后面介绍的应用包的信息展示，安装等等都会用到这个里面的文件。

创建完成后，cosmos会返回一个PackageRepositoryAddResponse给dcos UI, 当dcos UI发现repository add 成功的response，页面成功返回。

**仓库列表**

仓库添加成功后，dcos ui都会重新调用fetchRepositories的api来刷新当前的页面来返回最新的universe repository, 下面我们来就来看看这个api的调用:

当用户点击页面的时候，通过调用CosmosPackageAction.js::fetchRepositories去POST一个http request ,等待cosmos受理/repository/list的response, 代码如下：

fetchRepositories: function (type) {

RequestUtil.json({

contentType: getContentType('repository.list', 'request', 'v1'),

headers: {Accept: getContentType('repository.list', 'response', 'v1')},

method: 'POST',

url: `${Config.rootUrl}${Config.cosmosAPIPrefix}/repository/list`,

data: JSON.stringify({type}),

timeout: REQUEST\_TIMEOUT,

success: function (response) {

AppDispatcher.handleServerAction({

type: REQUEST\_COSMOS\_REPOSITORIES\_LIST\_SUCCESS,

data: response.repositories

});

},

error: function (xhr) {

AppDispatcher.handleServerAction({

type: REQUEST\_COSMOS\_REPOSITORIES\_LIST\_ERROR,

data: RequestUtil.getErrorFromXHR(xhr)

});

}

});

},

cosmos收到universe发过来的request后，通过调用PackageRepositoryListHandler::sourcesStorage.read()来获取当前的repository列表：

private[cosmos] final class PackageRepositoryListHandler(

sourcesStorage: PackageSourcesStorage

) extends EndpointHandler[PackageRepositoryListRequest, PackageRepositoryListResponse] {

override def apply(req: PackageRepositoryListRequest)(implicit

session: RequestSession

): Future[PackageRepositoryListResponse] = {

sourcesStorage.read().map(PackageRepositoryListResponse(\_))

}

}

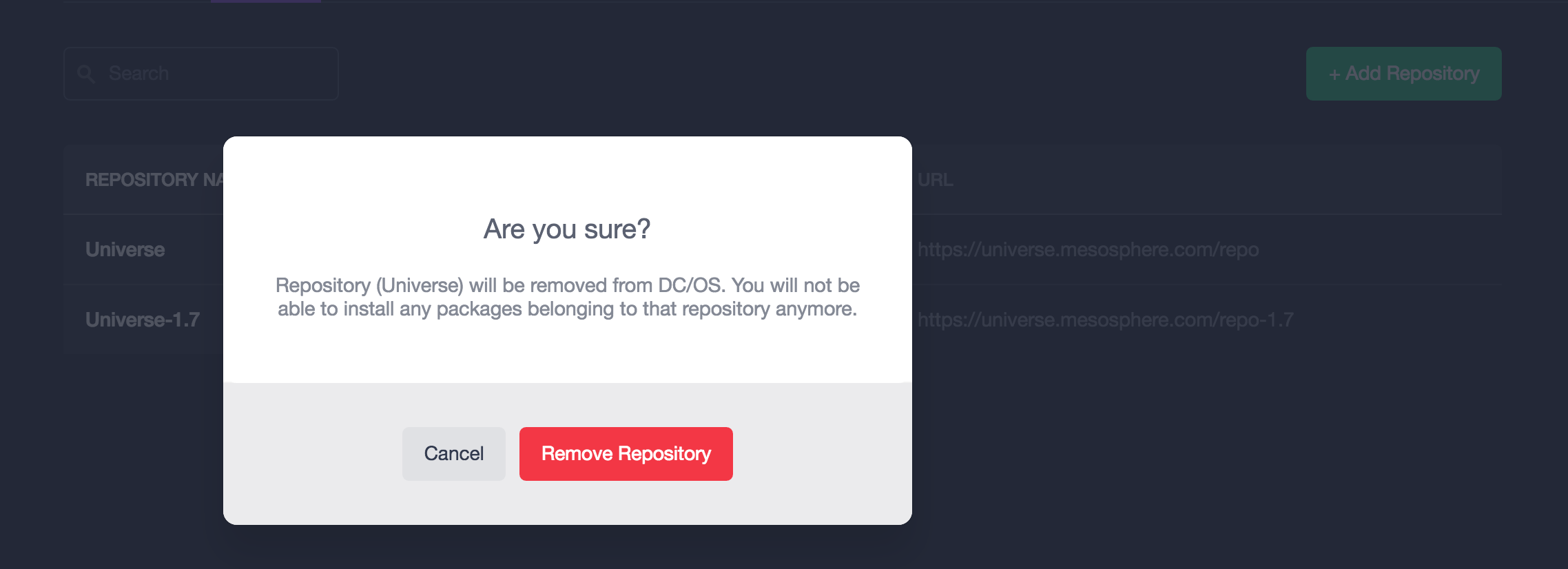
返回的PackageRepositoryListResponse是一个序列化的packageRepository类型，每一个packageRepository里面都有一个repositrory名字以及和其所对应的URI.

Universe收到cosmos的response后，RepositoriesTables.js会把这些数据展示到GUI上面。

**2.3 仓库删除**

用户除了能查看当前的仓库，创建新的仓库以外，当然还能删除已有的仓库。

在DCOS的GUI 上面，每一个仓库列表后面，都有一个remove的操作, 点击remove按钮，就会弹出对话框来确认参数操作：



点击Remove Repository, 操作开始。

同样的，dcos ui会调用CosmosPackageAction.js::deleteRepository的api来删除当前选择的universe repository, 请求会以post的方式发送给cosmos等待受理/repository/delete的response.

deleteRepository: function (name, uri) {

RequestUtil.json({

contentType: getContentType('repository.delete', 'request', 'v1'),

headers: {Accept: getContentType('repository.delete', 'response', 'v1')},

method: 'POST',

url: `${Config.rootUrl}${Config.cosmosAPIPrefix}/repository/delete`,

data: JSON.stringify({name, uri}),

timeout: REQUEST\_TIMEOUT,

success: function (response) {

AppDispatcher.handleServerAction({

type: REQUEST\_COSMOS\_REPOSITORY\_DELETE\_SUCCESS,

data: response,

name,

uri

});

},

error: function (xhr) {

AppDispatcher.handleServerAction({

type: REQUEST\_COSMOS\_REPOSITORY\_DELETE\_ERROR,

data: RequestUtil.getErrorFromXHR(xhr),

name,

uri

});

}

});

}

cosmos收到universe发过来的request后，通过调用PackageRepositoryDeleteHandler::sourcesStorage.delete()来删除这个universe repository：

private[cosmos] final class PackageRepositoryDeleteHandler(

sourcesStorage: PackageSourcesStorage

) extends EndpointHandler[PackageRepositoryDeleteRequest, PackageRepositoryDeleteResponse] {

import PackageRepositoryDeleteHandler.\_

override def apply(request: PackageRepositoryDeleteRequest)(implicit

session: RequestSession

): Future[PackageRepositoryDeleteResponse] = {

val nameOrUri = optionsToIor(request.name, request.uri).getOrElse(throw RepoNameOrUriMissing())

sourcesStorage.delete(nameOrUri).map { sources =>

PackageRepositoryDeleteResponse(sources)

}

}

删除完成后返回一个PackageRepository的List给dcos UI, 这个也就是最新的当前DC/OS系统中存在的universe repository了。

以上就是cosmos在DC/OS系统中对应用仓库的管理，包括仓库的查询，创建，删除这三部分，后期我们还会介绍软件包的应用部署等等。

作者简介：陈晖，供职于西安IBM, 主要从事云计算方面工作，涉及Openstack,Mesos等开源项目，在资源管理 ，资源调度等方面有比较丰富的经验 。