沙箱

Parcels

Parcels are an advanced feature of single-spa. Avoid using them until you have a better understanding of single-spa's registerApplication api. A single-spa parcel is a framework agnostic component. It is a chunk of functionality meant to be mounted manually by an application, without having to worry about which framework was used to implement the parcel or application. Parcels use similar methodology as registered applications but are mounted by a manual function call rather than the activity function. A parcel can be as large as an application or as small as a component and written in any language as long as it exports the correct lifecycle events. In a single-spa world, your SPA contains many registered applications and potentially many parcels. Typically we recommend you mount a parcel within the context of an application because the parcel will be unmounted with the application.

If you are only using one framework, it is recommended to prefer framework components (i.e., React, Vue, and Angular components) over single-spa parcels. This is because framework components interop easier with each other than when there is an intermediate layer of single-spa parcels. You may import components between registered applications via `import` statements. You should only create a single-spa parcel if you need it to work with multiple frameworks. ([More details](https://zh-hans.single-spa.js.org/docs/recommended-setup#in-browser-versus-build-time-modules))

沙箱是single-spa的一个先进特点。在您更好地理解single-spa的`registerApplication api`之前，不要使用它们。single-spa 沙箱是一个框架无关的组件。它是由应用程序手动安装的功能块，而不必担心使用哪个框架来实现沙箱或应用程序。沙箱使用与注册应用程序类似的方法，但是通过手动函数调用而不是激活函数来挂载。一个沙箱可以像应用程序一样大，也可以像组件一样小，只要它导出正确的生命周期事件，就可以用任何语言编写它。在single-spa世界中，您的 SPA 包含许多已注册的应用程序和潜在的许多沙箱。通常，我们建议您在应用程序上下文中挂载一个沙箱，因为这个沙箱将与应用程序一起卸载。

如果您只使用一个框架，建议您选择框架组件(即，React、Vue和Angular组件)。这是因为框架组件之间的交互操作比single-spa 沙箱的中间层更容易。您可以通过“import”语句在注册的应用程序之间导入组件。如果需要使用多个框架，应该只创建single-spa沙箱。([更多细节查看文档](https://zh-hans.single-spa.js.org/docs/recommended-setup#in-browser-versus-build-time-modules))

##### 快速上手的示例

[#](https://zh-hans.single-spa.js.org/docs/parcels-overview#quick-example)Quick Example

// The parcel implementation

const parcelConfig = {

bootstrap() {

// one time initialization

return Promise.resolve()

},

mount() {

// use a framework to create dom nodes and mount the parcel

return Promise.resolve()

},

unmount() {

// use a framework to unmount dom nodes and perform other cleanup

return Promise.resolve()

}

}

// How to mount the parcel

const domElement = document.getElementById('place-in-dom-to-mount-parcel')

const parcelProps = {domElement, customProp1: 'foo'}

const parcel = singleSpa.mountRootParcel(parcelConfig, parcelProps)

// The parcel is being mounted. We can wait for it to finish with the mountPromise.

parcel.mountPromise.then(() => {

console.log('finished mounting parcel!')

// If we want to re-render the parcel, we can call the update lifecycle method, which returns a promise

parcelProps.customProp1 = 'bar'

return parcel.update(parcelProps)

})

.then(() => {

// Call the unmount lifecycle when we need the parcel to unmount. This function also returns a promise

return parcel.unmount()

})

// 沙箱的实现

const parcelConfig = {

bootstrap() {

// one time initialization

return Promise.resolve()

},

mount() {

// 使用框架创建dom节点并挂载沙箱

return Promise.resolve()

},

unmount() {

// 使用框架卸载dom节点并执行其他清理

return Promise.resolve()

}

}

// 挂载沙箱的步骤

const domElement = document.getElementById('place-in-dom-to-mount-parcel')

const parcelProps = {domElement, customProp1: 'foo'}

const parcel = singleSpa.mountRootParcel(parcelConfig, parcelProps)

// 沙箱要完成挂载，我们需要等待它完成mountPromise。

parcel.mountPromise.then(() => {

console.log('完成挂载沙箱!')

// 如果我们要重新渲染包，我们可以调用update生命周期方法，它会返回一个promise

parcelProps.customProp1 = 'bar'

return parcel.update(parcelProps)

})

.then(() => {

// 在需要卸载沙箱时直接调用生命周期中的卸载方法。这个方法也返回一个promise

return parcel.unmount()

})

##### 沙箱配置

[#](https://zh-hans.single-spa.js.org/docs/parcels-overview#parcel-configuration)Parcel configuration

A parcel is just an object with 3 or 4 functions on it. When mounting a parcel, you can provide either the object itself or a loading function that asynchronously downloads the parcel object. Each function on a parcel object is a lifecycle method, which is a function that returns a promise. Parcels have three required lifecycle methods (bootstrap, mount, and unmount) and one optional lifecycle method (update). When implementing a parcel, it's strongly recommended that you use the [lifecycle helper methods](https://zh-hans.single-spa.js.org/docs/ecosystem#help-for-frameworks). An example of a parcel written in React would look like this:

// myParcel.js

import React from 'react'

import ReactDom from 'react-dom'

import singleSpaReact from 'single-spa-react'

import MyParcelComponent from './my-parcel-component.component.js'

export const MyParcel = singleSpaReact({

React,

ReactDom,

rootComponent: MyParcelComponent

})

// in this case singleSpaReact is taking our inputs and generating an object with the required lifecycles.

Then to use the parcel you just created all you need to do is use the `Parcel` component provided in [single-spa-react](https://zh-hans.single-spa.js.org/docs/single-spa-react.md#parcels)

// mycomponent.js

import Parcel from 'single-spa-react/parcel'

import MyParcel from './myparcel.js'

export class myComponent extends React.Component {

render () {

return (

<Parcel

config={MyParcel}

{ /\* optional props \*/ }

{ /\* and any extra props you want here \*/ }

/>

)

}

}

Note that in some cases the optional props are required [(see additional examples)](https://zh-hans.single-spa.js.org/docs/single-spa-react.md#examples).

一个沙箱就是一个有3或4个函数的对象。在挂载一个沙箱时，您可以提供对象本身，也可以提供一个异步下载沙箱对象的加载函数。沙箱对象上的每个函数都是一个生命周期方法，它是一个返回promise的函数。沙箱有三个必需的生命周期方法(初始化、挂载和卸载)和一个可选的生命周期方法(更新)。在实现一个沙箱时，强烈建议您使用[生命周期帮助助手](https://zh-hans.single-spa.js.org/docs/ecosystem#help-for-frameworks)方法。下面是用[React编写的沙箱示例](https://zh-hans.single-spa.js.org/docs/single-spa-react.md#parcels):

// myParcel.js

import React from 'react'

import ReactDom from 'react-dom'

import singleSpaReact from 'single-spa-react'

import MyParcelComponent from './my-parcel-component.component.js'

export const MyParcel = singleSpaReact({

React,

ReactDom,

rootComponent: MyParcelComponent

})

// 在这种情况下，singleSpaReact获取我们的输入并生成具有所需生命周期的对象。

然后，要使用您刚刚创建的沙箱，您所需要做的就是使用single-spa-react中提供的[沙箱(“parcel”)组件](https://zh-hans.single-spa.js.org/docs/single-spa-react.md#examples)

// mycomponent.js

import Parcel from 'single-spa-react/parcel'

import MyParcel from './myparcel.js'

export class myComponent extends React.Component {

render () {

return (

<Parcel

config={MyParcel}

{ /\* 配置可选的参数和属性 \*/ }

{ /\* 这里配置任何外部引入的参数和属性 \*/ }

/>

)

}

}

注意，在某些情况下，可选的参数和属性是必需的([参见其他示例](https://zh-hans.single-spa.js.org/docs/single-spa-react.md#examples))。

##### 沙箱生命周期

[#](https://zh-hans.single-spa.js.org/docs/parcels-overview#parcel-lifecycles)Parcel Lifecycles

Start with [applications](https://zh-hans.single-spa.js.org/docs/api#registered-application-lifecycle) to learn more about the functionality of single-spa's lifecycle methods.

从[应用程序注册章节](https://zh-hans.single-spa.js.org/docs/api#registered-application-lifecycle)开始，了解更多关于single-spa生命周期方法的功能。

###### 初始化

[#](https://zh-hans.single-spa.js.org/docs/parcels-overview#bootstrap)Bootstrap

This lifecycle function will be called once, right before the parcel is mounted for the first time.

function bootstrap(props) {

return Promise

.resolve()

.then(() => {

// This is where you do one-time initialization

console.log('bootstrapped!')

});

}

在第一次挂载沙箱之前，这个生命周期函数将被调用一次。

function bootstrap(props) {

return Promise

.resolve()

.then(() => {

// 这是进行一次性初始化的地方

console.log('初始化已完成!')

});

}

###### 挂载

[#](https://zh-hans.single-spa.js.org/docs/parcels-overview#mount)Mount

If the parcel is not mounted this lifecycle function is called when ever `mountParcel`

 is called. When called, this function should create DOM elements, DOM event listeners, etc. to render content to the user.

function mount(props) {

return Promise

.resolve()

.then(() => {

// This is where you tell a framework (e.g., React) to render some ui to the dom

console.log('mounted!')

});

}

如果沙箱没有挂载，则调用此生命周期函数并且“mountParcel”也被调用。当调用该函数时，该函数应该创建DOM元素、DOM事件监听器等，以便向用户呈现内容。

###### 卸载

[#](https://zh-hans.single-spa.js.org/docs/parcels-overview#unmount)Unmount

This lifecycle function will be called whenever the parcel is mounted and one of the following cases is true:

1、 `unmount()` is called

2、The parent parcel or application is unmounted

When called, this function should clean up all DOM elements, DOM event listeners, leaked memory, globals, observable subscriptions, etc. that were created at any point when the parcel was mounted.

function unmount(props) {

return Promise

.resolve()

.then(() => {

// This is where you tell a framework (e.g., React) to unrender some ui from the dom

console.log('unmounted!');

});

}

这个生命周期函数将在沙箱被挂载并且下列情况之一为真时被调用:

1、' unmount() '被调用

2、父沙箱或应用程序被卸载

当调用该函数时，该函数应该清理在挂载沙箱时创建的所有DOM元素、DOM事件监听器、泄漏内存、全局变量、可观察订阅等。

function unmount(props) {

return Promise

.resolve()

.then(() => {

// 这里是操作框架(例如，React)从dom中取消渲染某些UI的地方

console.log('unmounted!');

});

}

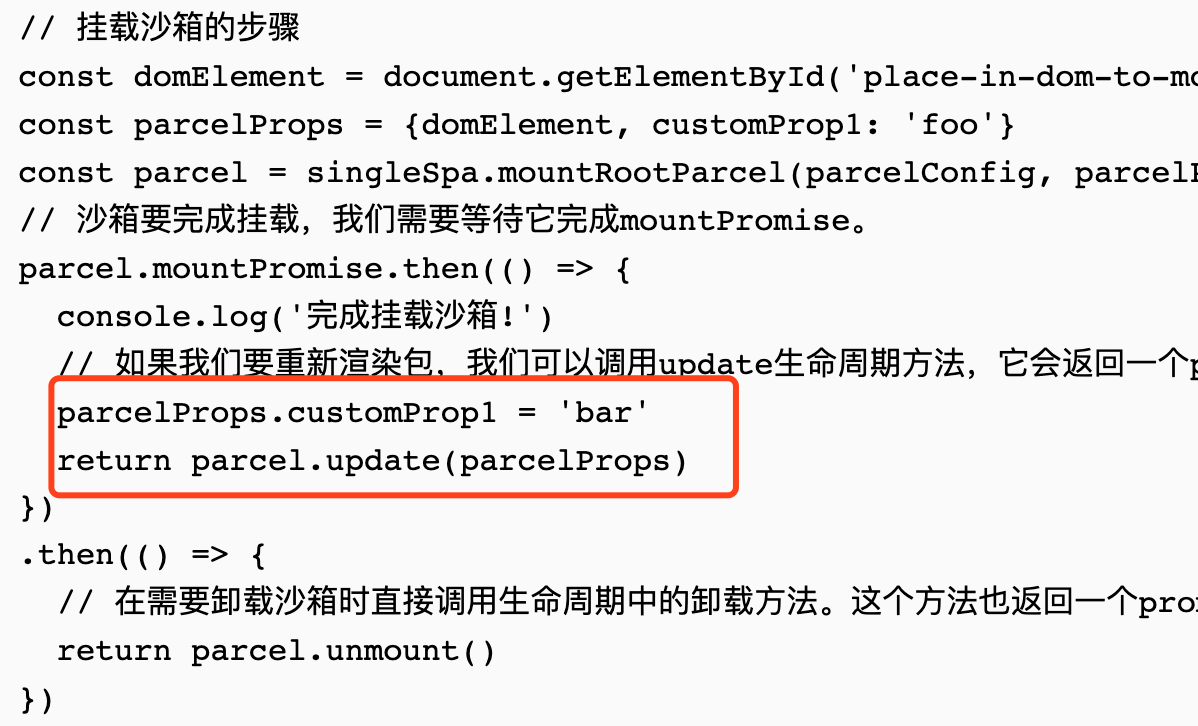
###### 更新(参数)

[#](https://zh-hans.single-spa.js.org/docs/parcels-overview#update-optional)Update (optional)

The update lifecycle function will be called whenever the user of the parcel calls `parcel.update()`. Single this lifecycle is optional, the user of a parcel needs to check whether the parcel has implemented the update lifecycle before attempting to make the call.

更新生命周期函数将在沙箱的用户调用“package .update()”时被调用。这个生命周期是可选的，沙箱的用户需要在尝试调用之前检查沙箱是否实现了更新生命周期。

示例：（补充：便于理解的示例）



##### 示例中用到的cases

[#](https://zh-hans.single-spa.js.org/docs/parcels-overview#example-use-cases)Example use cases

###### 模态框(Modals)

[#](https://zh-hans.single-spa.js.org/docs/parcels-overview#modals)Modals

`App1` handles everything related to contacts (highly cohesive) but somewhere in `App2` we need to create a contact. We could do any number of things to share the functionality between application 1 and 2:

* If both are written in the same framework we could export/import components.
* We could reimplement creating a contact (loss of cohesion)
* We could use single-spa parcels.

Exporting a parcel from `App1` that wraps the createContact modal component gives us the ability to share components and behavior across disparate frameworks, without losing application cohesion. App1 can export a modal as a single-spa parcel and App2 can import the parcel and use it easily. One major advantage is that in the below example the parcel/modal from App1 that is being used by App2 will also be unmounted, without unmounting/mounting of App1.

// App1

export const AddContactParcel = {

bootstrap: bootstrapFn,

mount: mountFn,

unmount: unmountFn,

}

// App2

// get the parcel configuration in this case I'm using systemJS and react

...

componentDidMount() {

SystemJS.import('App1').then(App1 => {

const domElement = document.body

App2MountProps.mountParcel(App1.AddContactParcel, {domElement})

})

}

...

' App1 '处理所有与联系人相关的内容(高度内聚)，但是在' App2 '的某个地方，我们需要创建一个联系人。我们可以做很多事情来共享应用1和2之间的功能:

1、如果两者都是在同一个框架中编写的，我们可以导出/导入组件。

2、我们可以重新实现创建一个contact(丧失内聚性)

3、我们可以使用single-spa 沙箱。

从包装了createContact模式组件的“App1”导出一个包，使我们能够跨不同的框架共享组件和行为，而不会丢失应用程序的内聚性。App1可以导出一个模态single-spa沙箱，App2可以导入沙箱并轻松使用它。一个主要的优点是，在下面的示例中，App2使用的来自App1的 parcel/modal 也将被卸载，而不需要卸载/安装App1。

// App1

export const AddContactParcel = {

bootstrap: bootstrapFn,

mount: mountFn,

unmount: unmountFn,

}

// App2

// 在本例中，我使用的是systemJS和react

...

componentDidMount() {

SystemJS.import('App1').then(App1 => {

const domElement = document.body

App2MountProps.mountParcel(App1.AddContactParcel, {domElement})

})

}

...

###### `mountRootParcel` vs `mountParcel`

[#](https://zh-hans.single-spa.js.org/docs/parcels-overview#mountrootparcel-vs-mountparcel) `mountRootParcel` vs `mountParcel`

Single spa exposes two APIs for working with parcels. These API's are differentiated primarily by the context in which the parcel is created and how to access the API's

|  |  |  |
| --- | --- | --- |
| context | singleSpa | application |
| unmount condition | manual only | manual + application unmount |
| api location | singleSpa named export | provided in lifecycle prop |

Single spa公开了两个处理沙箱的API。这些API主要由创建沙箱的上下文和如何访问API来区分

|  |  |  |
| --- | --- | --- |
| 上下文 | singleSpa | 应用程序 |
| 卸载条件 | 仅手册 | 手册+应用程序卸载函数 |
| 本地API | singleSpa 的 export 方式 | 暴露生命周期属性prop |

###### 我应该使用那种呢？

[#](https://zh-hans.single-spa.js.org/docs/parcels-overview#which-should-i-use)Which should I use?

In general we suggest using the application-aware `mountParcel`API. `mountParcel` allows you to treat the parcel just like a component inside your application without considering what framework it was written in and being forced to remember to call unmount.

一般来说，我们建议使用支持应用程序的 “mountParcel” API 。“mountParcel”允许你像对待你的应用程序中的一个组件一样对待这个沙箱，而不需要考虑它是在什么框架中编写的，也不需要记住调用unmount。

###### 如何获得“mountParcel” API?

[#](https://zh-hans.single-spa.js.org/docs/parcels-overview#how-do-i-get-the-mountparcel-api)How do I get the `mountParcel`API?

In order to keep the function contextually bound to an application it is provided to the application as a [lifecycle prop](https://zh-hans.single-spa.js.org/docs/building-applications#lifecyle-props). You will need to store and manage that function yourself in your application.

Example of storing the application specific `mountParcel`API:

// App1

let mountParcel

export const bootstrap = [

(props) => {

mountParcel = props.mountParcel

return Promise.resolve()

},

// more bootstrap lifecycles if necessary

]

...

note: some libraries (such as react) support a framework specific context that makes it easy to store/manage. In those cases we've written some helper methods to abstract away the need to manage and store the `mountParcel` method.

为了将函数上下文绑定到应用程序，将其作为生命周期支持提供给应用程序。您需要在应用程序中自己存储和管理该功能。

存储应用程序特定的' mountParcel ' api的例子:

// App1

let mountParcel

export const bootstrap = [

(props) => {

mountParcel = props.mountParcel

return Promise.resolve()

},

// more bootstrap lifecycles if necessary

]

...

注意事项:有些库(如react)支持特定框架的上下文，使其易于存储/管理。在这些情况下，我们编写了一些辅助方法来抽象出管理和存储“mountParcel”方法的需求。