路由接口说明

/appointment/views.py

all(): 获取用户所有预约信息

请求方法：GET

请求URL：/api/appointments

请求参数：无需参数

返回结果：返回当前用户所有预约信息的JSON格式列表。如果没有任何预约信息，则返回一个空列表。

add(): 创建预约

请求方法：POST

请求URL：/api/appointments

请求参数：

user\_id: 预约用户的ID（必须）

equipment\_id: 预约设备的ID（必须）

start\_time: 预约开始时间（必须）

end\_time: 预约结束时间（必须）

返回结果：如果创建成功，返回创建的预约信息的JSON格式；否则返回错误信息。

delete(): 删除预约

请求方法：POST

请求URL：/api/appointments/:id/delete （:id 为要删除的预约ID）

请求参数：无需参数

返回结果：如果删除成功，返回一个空JSON；否则返回错误信息。

edit(): 编辑预约

请求方法：POST

请求URL：/api/appointments/:id/edit （:id 为要编辑的预约ID）

请求参数：

user\_id: 预约用户的ID

equipment\_id: 预约设备的ID

start\_time: 预约开始时间

end\_time: 预约结束时间

返回结果：如果编辑成功，返回编辑后的预约信息的JSON格式；否则返回错误信息。

search\_by\_user(): 查询某个用户的所有预约信息

请求方法：GET

请求URL：/api/appointments/user/:id （:id 为要查询的用户ID）

请求参数：无需参数

返回结果：返回该用户的所有预约信息的JSON格式列表。如果该用户没有任何预约信息，则返回一个空列表。

search\_by\_equipment(): 根据设备ID查询当前已经预约的时间段

请求方法：GET

请求URL：/api/appointments/equipment/:id （:id 为要查询的设备ID）

请求参数：无需参数

返回结果：返回该设备已经预约的时间段的JSON格式列表。如果该设备在当前时间段没有任何预约信息，则返回一个空列表。

allow(): 显示每种设备在当天还可以预约的时间段

请求方法：GET

请求URL：/api/appointments/allow

请求参数：无需参数

返回结果：返回每种设备在当天还可以预约的时间段的JSON格式列表。如果某种设备在当天已经被预约满了，则不会包含在返回结果中。网IP的方式。

/collect/views.py

all\_award()：

请求方法：GET

请求URL：/all\_award

请求参数：无需参数

返回结果：返回当前用户所有获奖信息的JSON格式列表。如果没有任何获奖信息，则返回一个空列表。

commit()：

请求方法：POST

请求URL：/commit

请求参数：

name: 获奖名称（必须）

date: 获奖日期（必须，格式为"%Y-%m-%d"，例如："2022-01-01"）

level: 获奖等级（必须）

institution: 颁奖机构（必须）

user\_name: 获奖人姓名（必须）

返回结果：如果提交成功，返回一个包含 "msg" 字段的JSON；否则返回包含错误信息的JSON。

/equipment/views.py

/all接口：

请求方法：GET

请求URL：/all

请求参数：无

返回结果示例：

{

"msg": "获取成功",

"data": [

{

"id": 1,

"name": "设备1",

"location": "地点1",

"kind": "类型1",

"capacity": 100,

"image\_url": "http://example.com/image1.jpg"

},

{

"id": 2,

"name": "设备2",

"location": "地点2",

"kind": "类型2",

"capacity": 200,

"image\_url": "http://example.com/image2.jpg"

}

]

}

/add接口：

请求方法：POST

请求URL：/add

请求参数：

{

"name": "设备名称",

"location": "设备位置",

"kind": "设备类型",

"capacity": 100,

"image\_url": "http://example.com/image.jpg"

}

返回结果示例：

{

"msg": "添加成功",

"equipment\_id": 3,

"equipment\_name": "设备名称"

}

/delete接口：

请求方法：POST

请求URL：/delete

请求参数：

{

"equipment\_id": 3

}

返回结果示例：

{

"msg": "成功删除设备, 设备名: 设备名称"

}

/edit接口：

请求方法：POST

请求URL：/edit

请求参数：

{

"equipment\_id": 3,

"name": "新设备名称",

"location": "新设备位置",

"kind": "新设备类型",

"capacity": 150,

"image\_url": "http://example.com/new\_image.jpg"

}

返回结果示例：

{

"msg": "成功修改设备信息",

"id": 3,

"name": "新设备名称",

"location": "新设备位置",

"kind": "新设备类型",

"capacity": 150,

"image\_url": "http://example.com/new\_image.jpg"

}

/upload\_image接口：

请求方法：POST

请求URL：/upload\_image

请求参数：包含一个名为image的文件字段，值为上传的图片文件

返回结果示例：

{

"msg": "上传成功",

"image\_url": "http://example.com/uploaded\_image.jpg"

}

/upload\_inside\_image接口：

请求方法：POST

请求URL：/upload\_inside\_image

请求参数：包含一个名为image的文件字段，值为上传的图片文件

返回结果示例：

{

"msg": "上传成功",

"image\_url": "http://example.com/uploaded\_inside\_image.jpg"

}

/disable接口：

请求方法：POST

请求URL：/disable

请求参数：

{

"equipment\_id": 3

}

返回结果示例：

{

"msg": "成功禁用设备，设备名: 设备名称"

}

/enable接口：

请求方法：POST

请求URL：/enable

请求参数：

{

"equipment\_id": 3

}

返回结果示例：

{

"msg": "成功解除禁用设备，设备名: 设备名称"

}

/sign/views.py

/all获取所有设备信息接口：

请求方法：GET

请求URL：/all

请求参数：无

返回结果示例：

{

"msg": "获取成功",

"data": [

{

"id": 1,

"name": "设备1",

"location": "地点1",

"kind": "类型1",

"capacity": 100,

"image\_url": "http://example.com/image1.jpg"

},

{

"id": 2,

"name": "设备2",

"location": "地点2",

"kind": "类型2",

"capacity": 200,

"image\_url": "http://example.com/image2.jpg"

}

]

}

/add添加设备接口：

请求方法：POST

请求URL：/add

请求参数：

{

"name": "设备名称",

"location": "设备位置",

"kind": "设备类型",

"capacity": 100,

"image\_url": "http://example.com/image.jpg"

}

返回结果示例：

{

"msg": "添加成功",

"equipment\_id": 3,

"equipment\_name": "设备名称"

}

删除设备接口 (/delete)：

请求方法：POST

请求URL：/delete

请求参数：

{

"equipment\_id": 3

}

返回结果示例：

{

"msg": "成功删除设备, 设备名: 设备名称"

}

/edit修改设备信息接口：

请求方法：POST

请求URL：/edit

请求参数：

{

"equipment\_id": 3,

"name": "新设备名称",

"location": "新设备位置",

"kind": "新设备类型",

"capacity": 150,

"image\_url": "http://example.com/new\_image.jpg"

}

返回结果示例：

{

"msg": "成功修改设备信息",

"id": 3,

"name": "新设备名称",

"location": "新设备位置",

"kind": "新设备类型",

"capacity": 150,

"image\_url": "http://example.com/new\_image.jpg"

}

/upload\_image上传设备图片接口：

请求方法：POST

请求URL：/upload\_image

请求参数：包含一个名为image的文件字段，值为上传的图片文件

返回结果示例：

{

"msg": "上传成功",

"image\_url": "http://example.com/uploaded\_image.jpg"

}

上传设备内部图片接口 (/upload\_inside\_image)：

请求方法：POST

请求URL：/upload\_inside\_image

请求参数：包含一个名为image的文件字段，值为上传的图片文件

返回结果示例：

{

"msg": "上传成功",

"image\_url": "http://example.com/uploaded\_inside\_image.jpg"

}

/disable禁用设备接口：

请求方法：POST

请求URL：/disable

请求参数：

{

"equipment\_id": 3

}

返回结果示例：

{

"msg": "成功禁用设备，设备名: 设备名称"

}

/enable解除禁用设备接口：

请求方法：POST

请求URL：/enable

请求参数：

{

"equipment\_id": 3

}

返回结果示例：

{

"msg": "成功解除禁用设备，设备名: 设备名称"

}

/user/views.py

login() - 处理用户登录请求

请求方法：POST

请求URL：/user/login

请求参数：表单数据

返回结果：JSON格式的响应数据

register() - 处理用户注册请求

请求方法：POST

请求URL：/user/register

请求参数：表单数据

返回结果：JSON格式的响应数据

login\_out() - 处理退出登录请求

请求方法：GET

请求URL：/user/login\_out

请求参数：无

返回结果：JSON格式的响应数据

get\_image\_code() - 获取图片验证码

请求方法：GET

请求URL：/user/get\_image\_code

请求参数：无

返回结果：带有图片验证码的HTTP响应

send\_code\_by\_email() - 通过邮箱发送验证码

请求方法：POST

请求URL：/user/send\_code\_by\_email

请求参数：JSON数据，包含邮箱地址

返回结果：JSON格式的响应数据

reset\_password() - 找回密码（未登录的情况）

请求方法：POST

请求URL：/user/reset\_password

请求参数：表单数据

返回结果：JSON格式的响应数据

change\_password() - 修改密码（已登录的情况）

请求方法：POST

请求URL：/user/change\_password

请求参数：表单数据

返回结果：JSON格式的响应数据

change\_userinfo() - 修改用户信息（已登录的情况）

请求方法：POST

请求URL：/user/change\_userinfo

请求参数：表单数据

返回结果：JSON格式的响应数据

算法说明

canlendar.js实现了一个日历生成工具

包括了一个名为dyCalendar的对象，其中定义了一个draw方法。该方法接受一个option对象作为参数，用于指定生成日历的一些选项，比如目标位置、日期格式、是否高亮显示今天等。

在draw方法内部，首先根据选项中的年份和月份计算出该月的第一天是星期几，然后根据该信息创建出该月份的表格，并依次填充每个单元格，从1号开始到该月最后一天。同时，如果选项中指定了要高亮显示今天或者其他特定日期，也会在相应的单元格上添加CSS类名以实现高亮显示。

整个生成过程都是通过调用一系列辅助函数来完成的。除了绘制月份表格的函数外，还包括了一些用于处理日期、点击事件和CSS样式的函数。通过将这些功能函数封装在一个对象中，并提供一个简单的API来调用它们，可以使得用户能够灵活地定制自己需要的日历样式和功能。

header.js实现了一个简单的导航栏点击切换功能

它首先通过document.querySelectorAll('.nav>a')选择所有导航栏中的链接元素，并将它们放入一个NodeList对象中。然后，通过一个for循环遍历每个链接元素，并为它们绑定了一个onclick事件。当用户点击某个链接时，对应的事件处理函数就会被调用。在事件处理函数内部，首先通过document.getElementsByClassName('active')[0]找到当前处于激活状态的链接元素，然后移除其active类名，以便在接下来的操作中能够正确地添加到新的激活元素上。接着，通过event.target获取到当前被点击的链接元素，并给它添加active类名，以此将其标记为激活状态。

home.js用于创建日历的JavaScript库

它包括了一些函数来创建月份表格、绘制日历月份表格、获取日历详细信息、绘制日历以及处理用户点击事件的功能。

在这段代码中，作者使用了自执行函数来创建一个全局对象 dycalendar，并向这个对象添加了一些方法，比如 draw 方法用于绘制日历，getData 方法用于获取某一天的预约数据，以及一些辅助函数用于处理日历的绘制和用户交互。

news.js

首先，通过document.querySelector和document.querySelectorAll获取到左右按钮、小圆点和图片容器等DOM元素。然后，定义变量index和last用于记录当前显示的图片索引和上一张图片的索引。

position函数用于设置图片容器的left样式，使其根据当前索引显示对应的图片。

add函数用于增加索引，并判断是否需要循环到第一张图片。

desc函数用于减小索引，并判断是否需要循环到最后一张图片。

timer函数是一个定时器，用于每隔3秒自动切换图片。

dot函数用于设置当前小圆点的样式。接下来，给左右按钮添加点击事件监听器，在点击事件中调用相应的函数进行切换图片和更新样式，并清除定时器重新启动。最后，调用timer函数启动定时器。

script.js用于显示已预约的时间，并提供签到功能

showReservedTime函数用于显示已预约的时间。首先，它会清空原先的预约时间列表。然后，根据当前日期和时间，构造起始时间和结束时间的请求体。使用fetch函数向服务器发送POST请求，获取已预约的时间数据。然后，根据一定的规则对时间进行排序，包括是否过期、是否已签到等。排序完成后，将时间数据渲染到预约时间列表中。对于每个预约时间，创建一个tr元素，并填充相关信息，如设备名称、申请人姓名、开始时间、结束时间等。如果预约时间已过期，则显示"已过期"按钮，并禁用点击事件。如果已签到，则显示"已签到"按钮，并禁用点击事件。如果未审核，则显示"未审核"按钮，并禁用点击事件。如果可以签到，则显示"签到"按钮，并绑定点击事件。

handleSignIn函数用于处理签到逻辑。当用户点击"签到"按钮时，向服务器发送签到请求，并在成功后刷新页面并显示签到成功的提示。

init函数用于初始化页面。它禁用了打印功能，并在页面加载完成后调用showReservedTime函数显示已预约的时间。同时，还设置了定时器，每隔5分钟刷新一次预约时间列表。