对于这个奇怪的参数 path_regex = r"(?P<path>(?:(?:/[^/]+)+|/?))"
?P<path>表示接下来的是一个path，后边的括号中的内容是拿正则表达式匹配path的方式

先来看 services/contents/handlers.py 中的 ContentsHandler中的get方法说明

```
Return a model for a file or directory.
A directory model contains a list of models (without content)
of the files and directories it contains.
```

static/services/contents.js 的 Contents.prototype.get里有设置请求的type和format
services/contents/handlers.py 的get方法中，先获取请求的type和format (通过self.get_query_argument)，要获取目录下条目的话应该type=directory
通过self.contents_manager.get(path=path, type=type, format=format, content=content,)获得目录的model对象

base/handlers.py中有ContentsHandler的基类的基类IPythonHandler，其中contents_manager是一个property，通过self.settings['contents_manager']来获得contents_manager对象。那么self.settings是从哪里来的呢？我发现是在notebookapp.py里配置的settings['contents_manager']=contents_manager，contents_manager是FileContentsManager的一个实例，而FileContentsManager是ContentsManager的子类，实现了get方法
那么来看services/contents/filemanager.py中的FileContentsManager对象
它的get方法说明

```
Takes a path for an entity and returns its model

Parameters
----------
path : str
    the API path that describes the relative path for the target
content : bool
    Whether to include the contents in the reply
type : str, optional
    The requested type - 'file', 'notebook', or 'directory'.
    Will raise HTTPError 400 if the content doesn't match.
format : str, optional
    The requested format for file contents. 'text' or 'base64'.
    Ignored if this returns a notebook or directory model.

Returns
-------
model : dict
    the contents model. If content=True, returns the contents
    of the file or directory as well.
```

get方法中 if os.path.isdir(path)则model=self._dir_model(path, content=content)
那么来看_dir_model()方法，它里面最终实现了列出目录下各个文件的功能，并返回一个model，其中append的时候再次用到了get。。（这里是get一个文件，且不需要content）

```
if content:
    model['content'] = contents = []
    os_dir = self._get_os_path(path)
    for name in os.listdir(os_dir):
        try:
            os_path = os.path.join(os_dir, name)
        except UnicodeDecodeError as e:
            self.log.warning(
                "failed to decode filename '%s': %s", name, e)
            continue
        # skip over broken symlinks in listing
        if not os.path.exists(os_path):
            self.log.warning("%s doesn't exist", os_path)
            continue
        elif not os.path.isfile(os_path) and not os.path.isdir(os_path)
            self.log.debug("%s not a regular file", os_path)
            continue
        if self.should_list(name) and not is_hidden(os_path, self.root_
            contents.append(self.get(
                path='%s/%s' % (path, name),
                content=False)
            )

    model['format'] = 'json'

return model
```

那么如果要获取一个文件内容是怎么样的过程呢。以raw方式查看一个文件内容作为例子说明整个过程。

- 从html页面里看到点击一个item跳转到的链接如下，例如__init__.py的一个item（怎么动态生成的先不管了）
  - __init__.py
- 在edit/handlers.py里用EditorHandler类handle了这个链接，看它的get方法
  - 先判断错误情况，判断是否存在
  - self.write(self.render_template('edit.html', file_path=url_escape(path), basename=basename, page_title=basename + " (editing)",))
  - 看它的render_template参数里并没有文件内容，只传了path，说明是在edit.html里完成的显示文件内容，所以再看edit.html
- 看edit.html以及对应的js
  - 在这个container里显示的文件内容，当然，一开始还是什么都没有，要通过static_url("edit/js/built/main.min.js")来载入这些内容
  - 找到static/edit/js/main.js，发现实际上是在editor.js文件的Editor.prototype.load()里去载入文件内容的
  - load方法中有一句this.contents.get(this.file_path, {type: 'file', format: 'text'})
  - contents来自main.js中require的contents，来自static/services/contents.js
    - 其中的get方法设置url和type、format、content参数后，通过utils.promising_ajax异步访问处理完的最终url
    - 这个api_url生成的url还是api/content打头的，实际上跟文件路径列表的读取去向了同一个api
  - 又来到services/contents/handlers.py中的ContentsHandler
    - 通过self.contents_manager.get()获取文件内容
  - contents_manager实际上是services/contents/filemanager.py中类FileContentsManager的一个实例
    - get()方法中，对于raw文件的获取方式是这样的：model = self._file_model(path, content=content, format=format)
    - _file_mode()中，如果需要文件content，则会调用_read_file去读取
    - _read_file()来自fileio.py中的基类FileManagerMixin，除去一些错误判断，主要部分如下

```
with self.open(os_path, 'rb') as f:
    bcontent = f.read()

if format is None or format == 'text':
    # Try to interpret as unicode if format is unknown or if ur
    # was explicitly requested.
    try:
        return bcontent.decode('utf8'), 'text'
    except UnicodeError:
        if format == 'text':
            raise HTTPError(
                400,
                "%s is not UTF-8 encoded" % os_path,
                reason='bad format',
            )
return encodebytes(bcontent).decode('ascii'), 'base64'
```