上传：koa-multer

```
const upload = require('koa-multer')({dest: './public/uploads'});
router.post('/upload',
    upload.single('file'),
    async ctx => {
        console.log(ctx.req.file); // 注意数据存储在原始请求中
        // console.log(ctx.request.file);
        console.log(ctx.req.body);
        // console.log(ctx.request.body);
        ctx.body = '上传成功! ';
    })
```

校验：koa-bouncer

```
const bouncer = require('koa-bouncer')

router.post('/upload',
    bouncer.middleware({getBody: ctx => ctx.req.body}), // 指定数据来源
    async (ctx, next) => {
        // 校验逻辑
        try { // 验证不通过会抛出bouncer.ValidationError
            ctx.validateBody('uname')
                .trim() // 数据净化
                .isLength(6, 20, '请填写用户名')
            await next();
        } catch (error) {
            console.log(error);
            ctx.body = '校验错误: ' + error.message;
        }
    },
    async ctx => {
        // 净化过后的数据从ctx.vals获取
        console.log(ctx.vals);

        ctx.body = '上传成功! ';
    })
```

Session：koa-session

- 外部存储：必须实现三个异步方法get/set/destroy

```
const mysql = require('mysql')
const CREATE_SQL = `CREATE TABLE IF NOT EXISTS session_store(
                    session_id VARCHAR(255) NOT NULL,
                    expires BIGINT NULL,data TEXT NULL,
                    PRIMARY KEY(session_id), KEY session_store_expires(expires))`;
const GET_SQL = `select * from session_store where session_id=? and expires>?`;
```

```
const SET_SQL = `INSERT INTO session_store(session_id, expires, data)
                values(?,?,?) ON DUPLICATE KEY update expires=?,data=?`;

class MysqlStore {
    constructor(opts, pool) {
        // 定时清理
        // ...

        // 连接池判断
        if (pool) {
            this.connection = pool;
        } else {
            if (!opts || !opts.host || !opts.user || !opts.password) {
                throw new Error('需要额外配置数据库连接参数')
            }
            this.connection = mysql.createPool(opts);
        }

        // 执行建表逻辑
        this.query(CREATE_SQL)
    }

    // 适配器必须实现以下三个方法
    async get(sid, maxAge, {rolling}) {
        const rows = await this.query(GET_SQL, [sid, Date.now()]);
        let session = null;
        if (rows && rows[0] && rows[0].data) {
            session = JSON.parse(rows[0].data);
        }
        return session;
    }

    async set(sid, sess, maxAge, {rolling, changed}) {
        const expires = maxAge === 'session' ?
            new Date(Date.now()-86400000).getTime() :
            new Date(Date.now() + maxAge);
        const data = JSON.stringify(sess);
        const rows = await this.query(SET_SQL,
            [sid, expires, data, expires, data]);
        return rows;
    }

    async destroy(key) {
        // ...
    }

    query(sql, value) {
        return new Promise((resolve, reject) => {
            // resolve函数在异步操作成功时执行
            // reject函数在异步操作失败时执行
            this.connection.query(sql, value, (err, results) => {
                if (err) reject(err);

                else resolve(results);
```

```
            })
        })
    }
}

module.exports = MysqlStore
```

History API Fallback：

- koa2-connect-history-api-fallback
- 使用：

```
const fb = require('koa2-connect-history-api-fallback')
app.use(fb()) // 加载koa-static前面
```

下一步：

- egg.js：基于koa，底层实现很好
- think.js：ES6，MVC，底层实现很好

实时通信 - Socket.IO

1. 概述：浏览器和服务器之间实时的、双向的数据通信库
2. 组成：
    - 服务端：npm i socket.io -S
    - 客户端：npm i socket.io-client
3. 基础使用

```
// server
const io = require('socket.io')(server);
io.on('connection', socket => {
    console.log('用户来了');
    socket.on('disconnect', () => {
        console.log('用户走了');
    })
})

// client
<script src="/javascripts/socket.io.js"></script>
<script>
    const socket = io();
</script>
```

4. 客服
    - 一对一对话
        - 核心api：io.to(id).emit(event, msg)

作业

- 客服一对多对话:

  提示：服务器可以保存每个客服服务的用户数组
- 断开处理